清森学校
BEIJING QINGSEN SCHOOL

AP-CSA
Data structure and algorithms

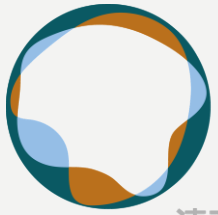
YING HUANG

D.2022

Data structure and algorithms

- A **data structure** is a named location that can be used to store and organize data.
- An **algorithm** is a collection of steps to solve a particular problem.

Program = Data Structure + Algorithm



Data structure and algorithms

Example 1

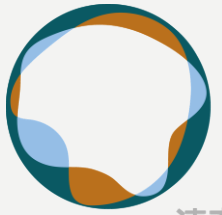
How to find the index of the target number in the array?

Target number is 42

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

Data Structure: Array, target value

Algorithms: methods to find the index



Algorithms

An algorithm is like a function

$$F(x) = y$$

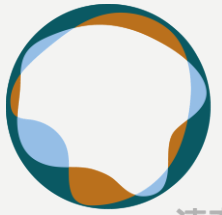
Algorithms

What makes a "good" algorithm?

- Correct (input and output)
- Efficient (processing)
- Readable/ Clear

- Why should we learn algorithms?

Algorithms is the Soul of Programming.



Data structure and algorithms

In AP-CSA

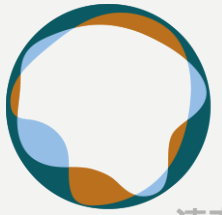
- Searching
  - ✓Sequential/Linear search
  - ✓Binary search
- Sorting
  - ✓Bubble Sort
  - ✓Selection Sort
  - ✓Insertion Sort
  - ✓Merge Sort (optional)

SEARCHING

SEARCHING

We can use traversals to search for individual elements in an Array/ ArrayList.

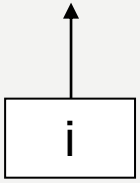
- Sequential Search/ Linear Search
- Binary Search

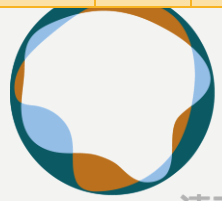


SEQUENTIAL SEARCH

- **Sequential Search/ Linear Search** checks each element in order until the target value or the end of the array/list is reached.
- **Sequential search** is the only method that can be used to find a value in unsorted data.
- Time Complexities: T = O(n)

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103





SEQUENTIAL SEARCH

Implement **sequential search** using arrays/list which returns the index of the target or -1 if it is not found.

Algorithms :

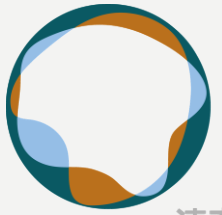
Step1 : Traverse through every value in array using loop

Step 2: Get value at index

Step 3: Check if target is value

Step 4: return index of value

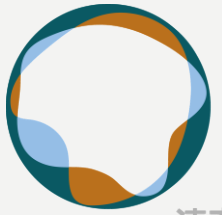
Step 5: If element is not in Array, return -1



SEQUENTIAL SEARCH

Code

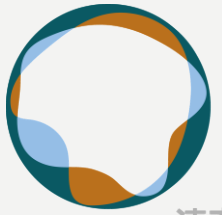
```
public int sequentialSearch(int[] array, int target){
    for(int i = 0; i < array.length; i++){
        if(array[i] == target)
            return i;
    }
    // target not in array
    return -1;
}
```



SEQUENTIAL SEARCH

Pros and Cons of Sequential Search:

- Sequential Search is fairly easy to implement and understand.
- As the size of the data increase, however, the longer Linear Search takes to complete.



SEQUENTIAL SEARCH

Example 1:

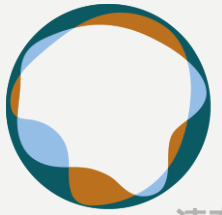
Which will cause the *shortest* execution of a sequential search looking for a value in an array of integers?

A. The value is the first one in the array

B. The value is in the middle of the array

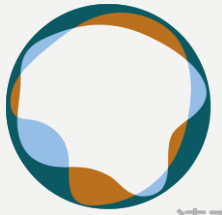
C. The value is the last one in the array

D. The value isn't in the array



BINARY SEARCH

- **Binary Search** can only be used if the data is sorted.
- **Binary Search** compares a target value to the value in the middle of a range of indices. If the value isn't found it looks again in either the left or right half of the current range.
- Each time through the loop it eliminates half the values in the search area until either the value is found or there is no more data to look at.
- Time complexity: T = O(Log n).



BINARY SEARCH

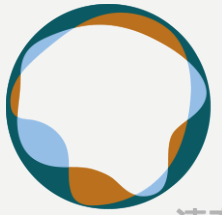
Searching the array below for the value 42:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

left

mid

right



BINARY SEARCH

Algorithms:

Step 1: Set left to 0 and right to length -1

Step 2: Compare left and right (left <= right) by while loop

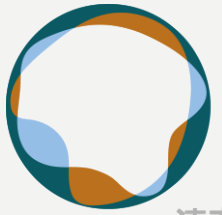
Step 3: Set mid = (left+right)/2

Step 4: If the target > midValue -> left = mid + 1

Step 5: If the target < midValue -> right = mid -1

Step 6: If the target = midValue -> return mid

Step 7: can not find, return -1 outside of while.



BINARY SEARCH

Code:

```
public static int binarySearch(int[] elements, int target) {
    int left = 0;
    int right = elements.length - 1;
    while (left <= right) {
        int middle = (left + right) / 2;
        if (target < elements[middle])   right = middle - 1;
        else if (target > elements[middle])   left = middle + 1;
        else return middle;
    }
    return -1;
}
```

BINARY SEARCH

Example 2:

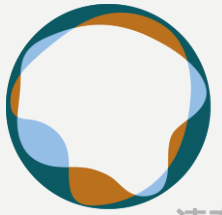
Which will cause the shortest execution of a binary search looking for a value in an array of integers?

A. The value is the first one in the array

B. The value is in the middle of the array

C. The value is the last one in the array

D. The value isn't in the array



BINARY SEARCH

Example 3:

Which of the following conditions must be true in order to search for a value using binary search?

I. The values in the array must be integers.

II. The values in the array must be in sorted order.

III. The array must not contain duplicate values.

A. I only

B. I and II

C. II only

D. II and III

BINARY SEARCH

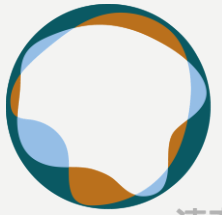
Example 4:

How many times would the while loop execute if you first do int[] arr = {2, 10, 23, 31, 55, 86} and then call binarySearch(arr,55)?

A. 2

B. 1

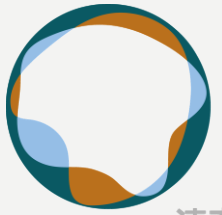
C. 3



HOMEWORK 7.4.8

correctlyFormatted

- This method returns true if all of the data in the list is formatted correctly.
- Correctly formatted names are made up of a first name and a last name, separated by a single space.
- Both the first and last names should start with an uppercase letter.



HOMEWORK 7.4.8

```
public static boolean correctlyFormatted(ArrayList<String> people){
    for(int i = 0; i < people.size(); i++) {
        System.out.println(people.get(i));
        if(people.get(i).length() < 3) {return false;}
        int spaceCount = 0;
        for(int j = 0; j < people.get(i).length(); j++){
            char ch = people.get(i).charAt(j);/
            if(j == 0 && (ch > 'Z' || ch < 'A')) {
                return false;
            }
            if(people.get(i).charAt(j) == ' ') {
                spaceCount++;
                if(spaceCount == 2) {
                    return false;
                }
                if(j == people.get(i).length() - 1) {
                    return false;
                }
                char cur = people.get(i).charAt(j + 1);
                if (cur > 'Z' || cur < 'A') {return false;}
            }
        }
        if(spaceCount == 0)  return false;
    }
    return true;
}
```

HOMEWORK 7.4.8

```
public static boolean correctlyFormatted(ArrayList<String> people){
    boolean yes = false;
    for (int i = 0; i < people.size(); i++){
        String[] splitN = people.get(i).split(" ");
        if (splitN.length != 2){
            return false;
        }
        else if (splitN.length == 2){
                if (splitN[0].substring(0,1).equals(splitN[0].substring(0,1).toUpperCase())
                &&
                splitN[1].substring(0,1).equals(splitN[1].substring(0,1).toUpperCase())){
                yes = true;
                }
                else{
                    return false;
                }
        }
    }
    return yes;
}
```

HOMEWORK 7.4.8

```
public static boolean correctlyFormatted(ArrayList<String> people)
{
    //travesering -loop
    for(int i = 0; i< people.size();i++){
        String name = people.get(i);
        String uppCase ="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        // whether it cantains space
        if(!name.contains(" ")) return false;

        // name-> Yumo Ouyang; name.substring(0,1) -> Y
        if(!uppCase.contains(name.substring(0,1))) return false;

        // last name
        int index = name.indexOf(" ");
        if(!uppCase.contains(name.substring(index+1,index+2))) return false;

    }

    return true;
}
```

HOMEWORK 7.4.8

```
34
35      public static boolean correctlyFormatted(ArrayList<String> bruh){
36          String checkString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
37          String oneOfThem;
38          int doubleCheck;
39
40          for(int i=0; i<bruh.size(); i++)
41          {
42              oneOfThem = bruh.get(i);
43              doubleCheck = oneOfThem.indexOf(" ");
44
45              if((doubleCheck == -1)||(doubleCheck==oneOfThem.length()-1)
46              ||(checkString.contains(oneOfThem.substring(0,1)) == false)
47              ||(checkString.contains(oneOfThem.substring(doubleCheck+1, doubleCheck+2)) == false))
48              {
49                  return false;
50
51              }
52
53
54          }
55          return true;
56      }
57  }
```

HOMEWORK 7.4.8

```
public static boolean correctlyFormatted(ArrayList<String> people){
    String space =" ";
    for(int i=0; i<people.size();i++){
        String name = people.get(i);
        if(!name.contains(space)){
            return false;
        }
        else
    {
        String firstName = name.substring(0,1);
        if( firstName != firstName.toUpperCase()){
            return false;
        }
        String lastName = name.split(" ")[1].substring(0,1);
        if(lastName !=lastName.toUpperCase() ){
            return false;
        }
    }

     }
    return true;
}
```

SORTING

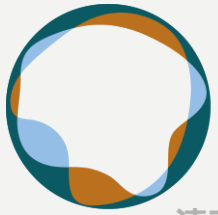
SORTING

When data is disorganized, it can be hard to find values easily:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	22	18	12	-4	27	30	36	50	7	68	91	56	2	85	42	98	25

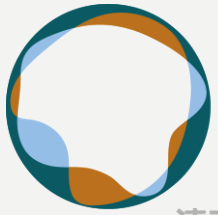
Organizing, or sorting data, can make it easier to search through:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	12	18	22	25	27	30	36	42	50	56	68	85	91	98



SORTING

- **sorting**: Rearranging the values in an array or collection into a specific order (usually into their "natural ordering").
  - one of the fundamental problems in computer science
  - can be solved in many ways:
    - there are many sorting algorithms
    - some are faster/slower than others
    - some use more/less memory than others
    - some work better with specific kinds of data
    - some can utilize multiple computers / processors, ...
  - *comparison-based sorting* : determining order by comparing pairs of elements:
    - `<`, `>`, `compareTo`, ...



SORTING ALGORITHMS

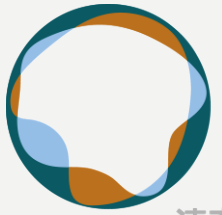
Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(k)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n + k)
Bucket Sort	O(n + k)	O(n + k)	O(n^2)	O(n)

SORTING ALGORITHMS

There are many sorting algorithms.

Wikipedia lists over 40 sorting algorithms. The following three sorting algorithm will be on the AP exam.

- **selection sort**: look for the smallest element, swap with first element. Look for the second smallest, swap with second element, etc…
- **insertion sort**: build an increasingly large sorted front portion of array.
- **merge sort**: recursively divide the array in half and sort it. Merge sort will be discussed in Unit 10.



SELECTION SORT

Selection Sort: Sorts an array by repeatedly finding the minimum value, and moving it to the front of the array.

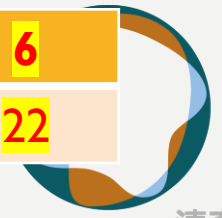
Index	0	1	2	3	4	5	6
value	22	31	-4	12	23	9	15

Index	0	1	2	3	4	5	6
value	-4	31	22	12	23	9	15

Index	0	1	2	3	4	5	6
value	-4	9	22	12	23	31	15

Index	0	1	2	3	4	5	6
value	-4	9	12	22	23	31	15

Index	0	1	2	3	4	5	6
value	-4	9	12	15	23	31	22

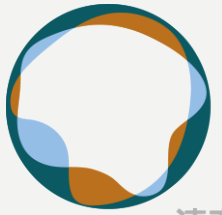


SELECTION SORT

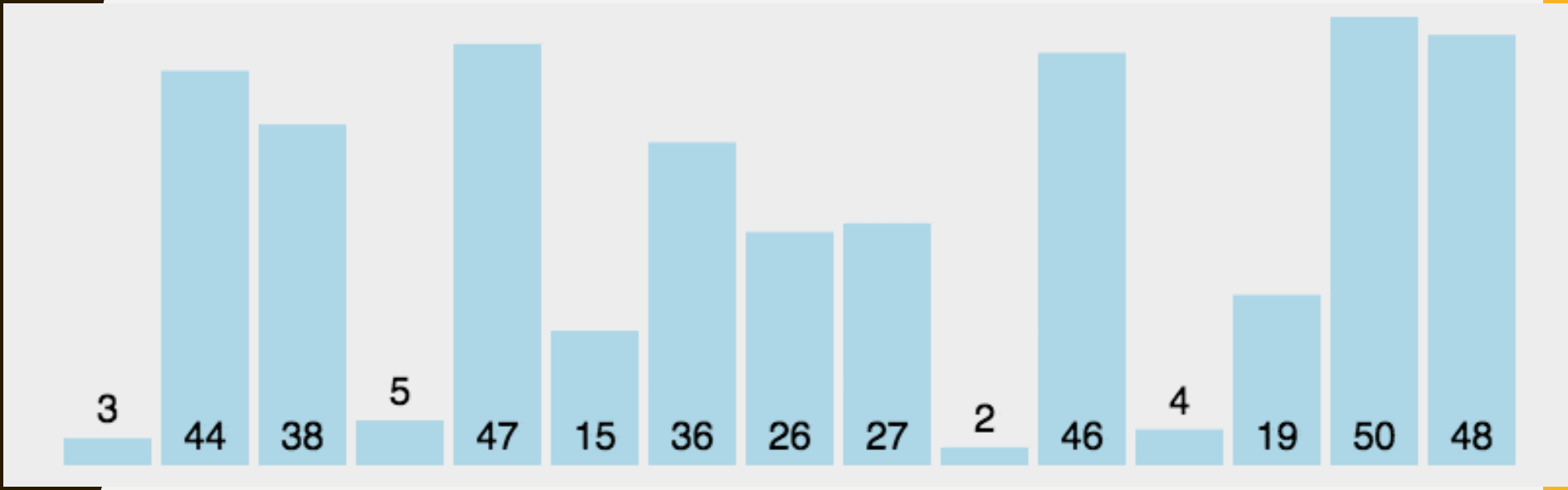
Index	0	1	2	3	4	5	6
value	-4	9	12	15	22	31	23

Index	0	1	2	3	4	5	6
value	-4	9	12	15	22	23	31

How can we implement this?



SORTING ALGORITHMS

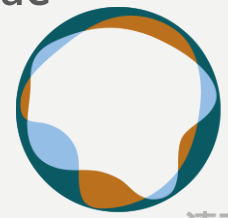


清森学校
BEIJING QINGSEN SCHOOL

SELECTION SORT

Algorithms – Pseudo code

1. Traverse each index up to the second to last element;
2. Find the minimum in the rest of the list;
   - ① Set current index to minimum
   - ② Traverse from current index to end of list
   - ③ If statement to determine which is minimum
3. Swap the index and minIndex
   - ① Create temporary variable to store current index value
   - ② Make current index value the minIndex value
   - ③ Make minIndex value the temporary variable value



SELECTION SORT

Code:

```
public static void selectionSort(int[] list){
    //Traverse each index up to the second to last element;
    for(int i=0; i< list.length-1; i++){
        //Find the minimum in the rest of the list;
        int minIndex = i;
        for(int j = i+1; j<list.length;j++){
            if(list[minIndex]>list[j]){
                minIndex=j;
            }
        }
        //Swap the index and minIndex
        if(i!=minIndex){
            int temp = list[i];
            list[i]=list[minIndex];
            list[minIndex]=temp;
        }
    }
```

CLASS WORK

What are errors for these code?

```
public static ArrayList<Integer> selectionSort1( ArrayList <Integer> arr){
    for(int i = 1;i<arr.size()-1;i++){
        int current = arr.get(i);
        int mini = current;
        int index = 0;
        for(int a = i+1;a<arr.size();a++){
            if(current>arr.get(a)){
                current = arr.get(a);
                index = a;
            }
        }
        int temp = current;
        int bigger = arr.get(i);
        arr.set(i-1,temp);
        arr.set(index,bigger);
    }
    return arr;
}
```

CLASS WORK

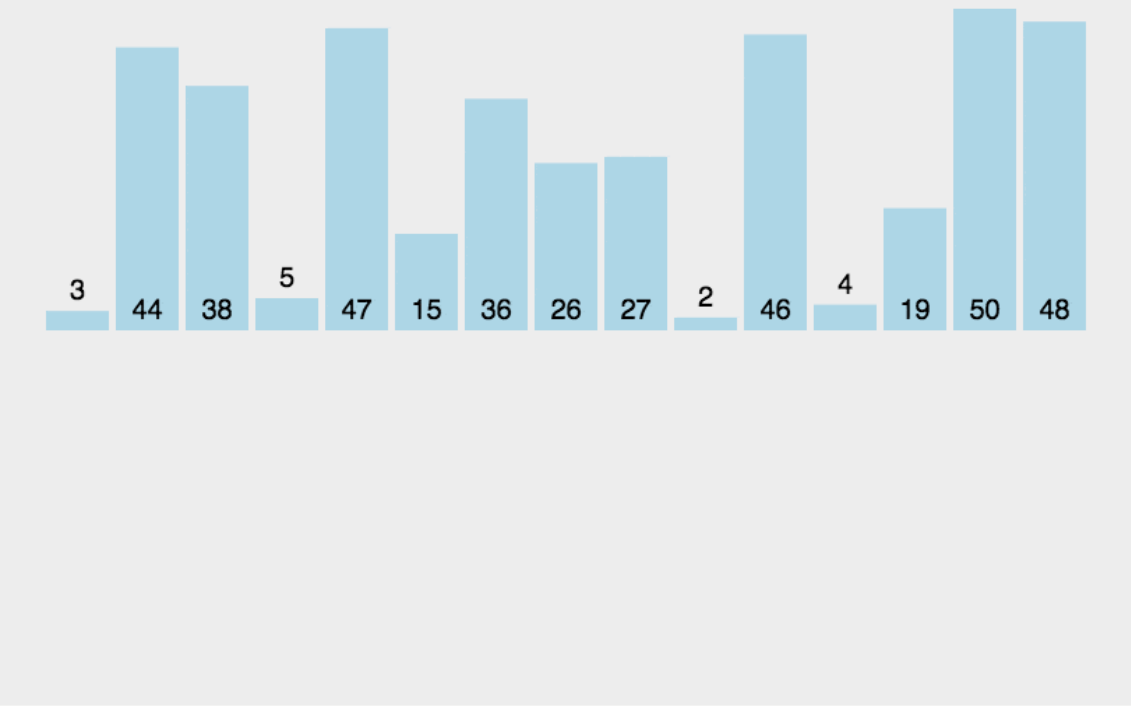
What are errors for these code?

```
public static int[] selectionSort2(int[] list)
{
    for(int i =0; i < list.length-1; i++)
    {
        int min = list[i];
        for(int j = i; j < list.length; j++)
        {
            if(list[j] < min)
            {
                int now = list[i];
                list[i] = list[j];
                list[j] = now;
            }
        }
    }
    return list;
}
```

清森学校
BEIJING QINGSEN SCHOOL

INSERTION SORT

Insertion sort sorts an array by sorting each element compared to the elements already sorted to their left.



Index	0	1	2	3	4
value	22	31	-4	12	23

Index	0	1	2	3	4
value	22	31	-4	12	23

Index	0	1	2	3	4
value	22	-4	31	12	23

Index	0	1	2	3	4
value	-4	22	31	12	23

Index	0	1	2	3	4
value	-4	22	12	31	23

Index	0	1	2	3	4
value	-4	12	22	31	23

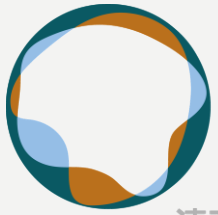
Index	0	1	2	3	4
value	-4	12	22	23	31

INSERTION SORT

How can we implement this?

Algorithms – Pseudo code

1. Traverse each element starting from index 1
2. Traverse sorted elements to find current element position
   - ① Set current value = list[index];
   - ② Set leftIndex;
   - ③ Use while-loop to determine that current value is less than the left numbers and set inbounds;
3. Shift sorted elements to place current element.
   1. Shift the value at the leftIndex to the right one place (in while- loop)
   2. Put the current value in the proper location (outside while-loop)



INSERTION SORT

Code:

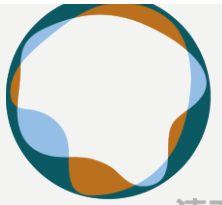
```
public static void insertionSort(int[] list)
{
    for (int i = 1; i < list.length; i++)
    {
        int curr = list[i];
        int leftIndex = i;
        while (leftIndex  > 0 && curr < list[leftIndex - 1])
        {
            list[leftIndex] = list[leftIndex - 1];
            leftIndex--;
        }
        list[leftIndex] = curr;
    }
}
```

INSERTION SORT

Code for ArrayList:

```
public static void insertionSort(ArrayList<Integer> list){
    for(int i = 1; i < list.size(); i++){
        int current = list.remove(i); // removes & returns
        int index = i;
        while(index > 0 && current < list.get(index-1))
            index--;
        list.add(index, current);
    }

}
```

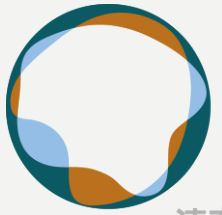


BIG O TIME COMPLEXITY

Question:

When resolving a computer-related problem, there will frequently be more than just one solution.

How will we compare these solution/ logic/ algorithms?

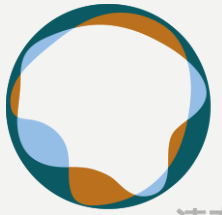


THE COMPLEXITY OF AN ALGORITHM

The complexity of an algorithm is the amount of resources (elementary operations or loop iterations) required for running it.

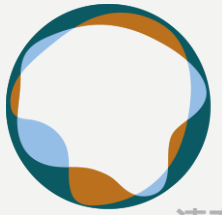
(lower the complexity = faster algorithm)

- Time Complexity (Big O notation)
- Space Complexity



BIG O TIME COMPLEXITY

- **Big O** Notation is a way to represent how long an algorithm will take to execute.
- **Big O** is the relationship runtime complexity of algorithms with the size of input data.
- **Big O** notation: $T = O(g(n))$
  - T represents the computing time of some algorithms.
  - g(n) represents a known standard function.
  - n represents the size of input data



BIG O - O(1)

- O(1): Constant time complexity will always take same amount of time to be executed.
- Example:

```
int[] array ={1,3,2,3,1,2,4,1,4,2,2,1,1};
```

```
if(i!=minIndex){
    int temp = list[i];
    list[i]=list[minIndex];
    list[minIndex]=temp;
```

BIG O - O(n)

O(n) - Linear time complexity

An algorithm has a linear time complexity if the time to execute the algorithm is directly proportional to the input size n.

Example: (sequential Search)

```java
public int sequentialSearch(int[] array, int target){
    for(int i = 0; i < array.length; i++){
        if(array[i] == target)
            return i;
    }
    // target not in array
    return -1;
}
```

BIG O - $O(n^2)$

$O(n^2)$ - Quadratic time complexity

An algorithm has quadratic time complexity if the time to execution it is proportional to the square of the input size.

Example:

(selection Sort)

```java
public static void selectionSort(int[] list){
    //Traverse each index up to the second to last element;
    for(int i=0; i< list.length-1; i++){
        //Find the minimum in the rest of the list;
        int minIndex = i;
        for(int j = i+1; j<list.length;j++){
            if(list[minIndex]>list[j]){
                minIndex=j;
            }
        }
        //Swap the index and minIndex
        if(i!=minIndex){
            int temp = list[i];
            list[i]=list[minIndex];
            list[minIndex]=temp;
        }
    }
```

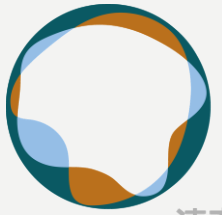
BIG O - O(log^n)

O(n^2) - Quadratic time complexity

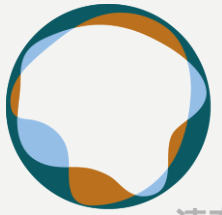
An algorithm has logarithmic time complexity if the time it takes to run the algorithm is proportional to the logarithm of the input size n.

Example: (binary searching)

```
public static int binarySearch(int[] elements, int target) {
    int left = 0;
    int right = elements.length - 1;
    while (left <= right) {
        int middle = (left + right) / 2;
        if (target < elements[middle])   right = middle - 1;
        else if (target > elements[middle])   left = middle + 1;
        else return middle;
    }
    return -1;
}
```



HOW ABOUT O($n \times log^n$)

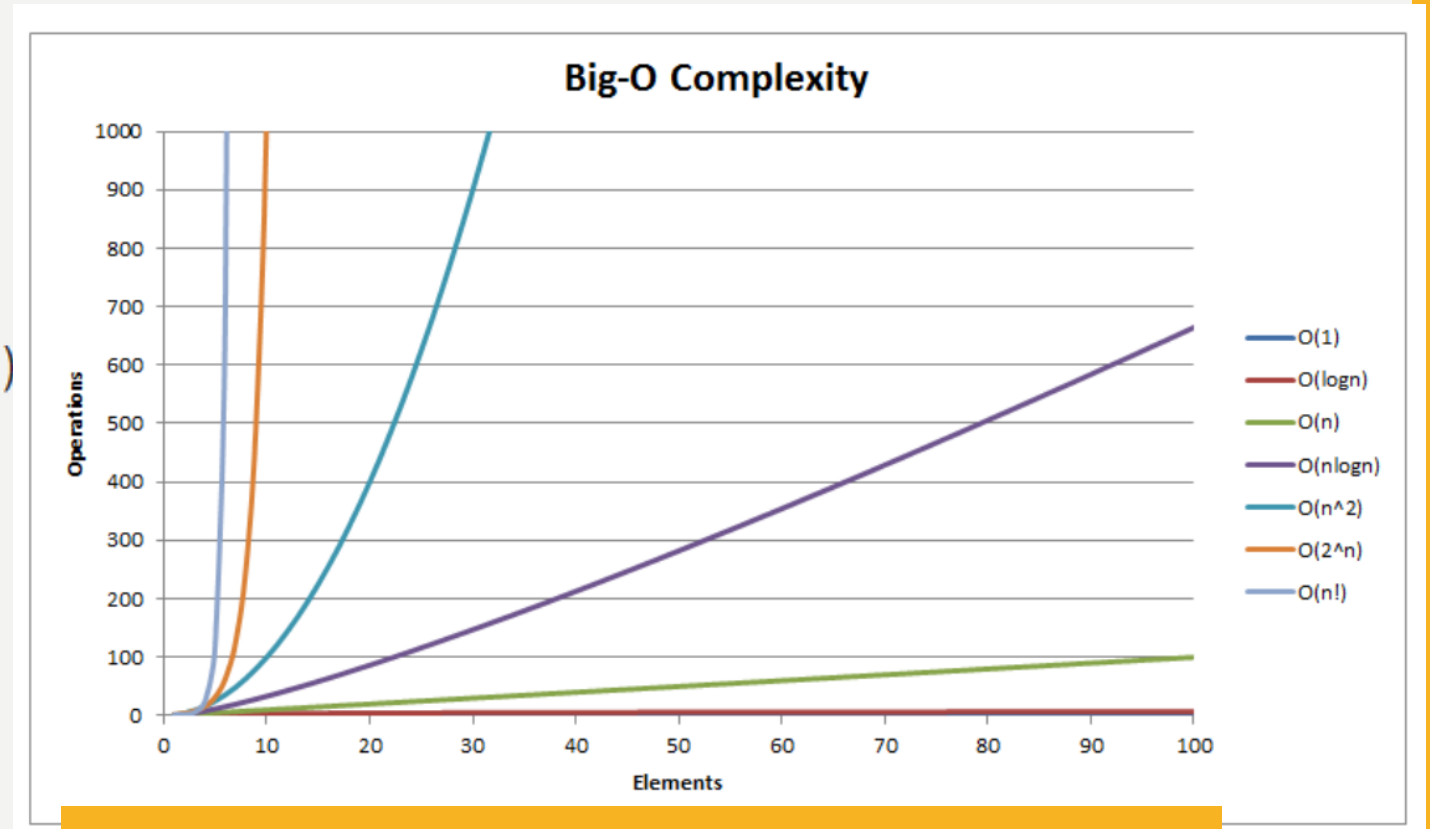


清森学校
BEIJING QINGSEN SCHOOL

BIG O

- Some of the lists of common computing times of algorithms in order of performance are as follows:

- O (1)
- O (log n)
- O (n)
- O (nlog n)
- O (n^2)
- O (n^3)
- O (2^n)



lower the complexity = faster algorithm

COMPLEXITY OF ALGORITHMS

Algorithm	Time Complexity		
	Best	Average	Worst
Sequential Search	O(1)	O(n)	O(n)
Binary Search	O(1)	$O(\log^n)$	$O(\log^n)$
Selection Search	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Search	O(n)	$O(n^2)$	$O(n^2)$

COMPLEXITY OF ALGORITHMS

Insertion and Selection Sort efficiency depends on how sorted the list order is at the start of the sort.

- Ascending order/ almost sorted:

  Insertion Sort has a lower execution count than **Selection Sort**, because the while – loop doesn't execute.

- Reverse order(Worst Case):

  Selection Sort has a lower execution count because it only needs swap two values, while **Insertion Sort** has to shift every single value.

COMPLEXITY OF ALGORITHMS

Example 5:

How many times as a function of n does the computation x++ executed?

```
int x = 0;
for(int i = 0; i < n; i++){
  x++;
}
```

Answer: n(linear function of n)

COMPLEXITY OF ALGORITHMS

Example 6:

How many times as a function of n does the computation x++ executed?

```
int x = 0;
for(int i = 0; i < n; i++){
  x++;
}

for(int j = 0; j < n; j++){
  x++;
}
```

Answer: 2n

COMPLEXITY OF ALGORITHMS

Example 7:

How many times as a function of n does the computation x++ executed?

```
int x = 0;
for(int i = 0; i < n; i++){
  for(int j = 0; j < n; j++){
     x++;
  }
}
```

Answer: n^2

COMPLEXITY OF ALGORITHMS

Example 8:

How many times as a function of n does the computation x++ executed?

```
int x = 1;
while((int)(Math.pow(2, x)) <= n){
  x++;
}
```

Answer: log_2^n

SEARCHING AND SORTING

Example 9:

Consider the binarySearch method below. How many times would the while loop execute if you first do int[] arr = {2, 10, 23, 31, 55, 86} and then call binarySearch(arr,2)?

A. 1

B. 2

C. 3

ANS: B

```
public static int binarySearch(int[] elements, int target) {
   int left = 0;
   int right = elements.length - 1;
   while (left <= right)
   {
      int middle = (left + right) / 2;
      if (target < elements[middle])
      {
         right = middle - 1;
      }
      else if (target > elements[middle])
      {
         left = middle + 1;
      }
      else {
         return middle;
      }
    }
    return -1;
}
```

SEARCHING AND SORTING

Example 10:

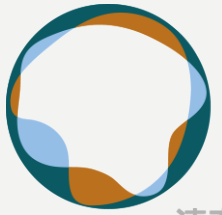
Under what condition will an ascending insertion sort execute the slowest?

A. If the data is already sorted in ascending order

B. If the data is already sorted in descending order

C. It will always take the same amount of time to execute

ANS: B



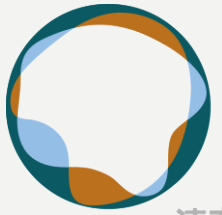
SEARCHING AND SORTING

Example 11:

Which of the following correctly shows the iterations of an ascending (from left to right) insertion sort on an array with the following elements: {7,3,8,5,2}?

A. {3,7,8,5,2}, {3,7,8,5,2}, {3,5,7,8,2}, {2,3,5,7,8}
B. {2,3,8,5,7}, {2,3,8,5,7}, {2,3,5,8,7}, {2,3,5,7,8}
C. {3,7,8,5,2}, {3,5,7,8,2}, {2,3,5,7,8}
D. {2,3,8,5,7}, {2,3,5,8,7}, {2,3,5,7,8}
E. {2,7,3,8,5}, {2,3,7,8,5}, {2,3,5,7,8}

ANS: A



SEARCHING AND SORTING

Example 12:

What would test return if a = {1,2,3,4} and v = 3?

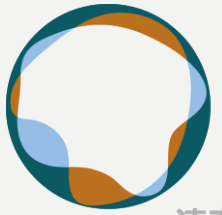
A. 0

B. 1

C. 2

D. The code will not compile

ANS: D



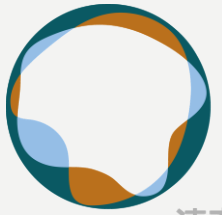
SEARCHING AND SORTING

Example 13:

What is printed when the following main method is executed?

```
public static void main(String[] args)
{
    int count = 0;
    int[] numbers = {-5,4,-5,3,-2,-4};
    for (int j = 0; j < numbers.length; j++)
    {
        if(numbers[j] < 0 && numbers[j] % 2 != 0)
        {
            count++;
        }
    }
System.out.println(count);
}
```

ANS:
2



SEARCHING AND SORTING

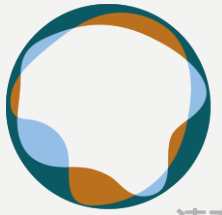
Example 14:

What is printed when the following main method is executed?

```
public static void main(String[] args)
{
    int[] arr = {8,7,7,3,4,1};
    for (int i = 0; i < arr.length; i++)
    {
        if(arr[i] % 2 == 0)
        {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
        }
    }
    for (int t = 0; t < arr.length; t++)
    {
        System.out.print((arr[t]) + ",");
    }
}
```

ANS:

4,7,7,3,8,1



SEARCHING AND SORTING

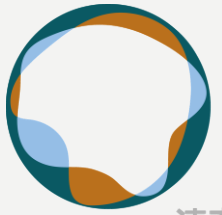
Example 15:

What is printed when the following main method is executed?

```
public static void main(String[] args)
{
    int[] arr = {5,3,2,9,3,4};
    for (int i = 0; i < arr.length; i++)
    {
        if(check(arr[i]))
        {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
        }
    }
    for (int t = 0; t < arr.length; t++)
    {
        System.out.print((arr[t]) + ",");
    }
}
```

```
private static boolean check(int n)
{
    for(int i = 2; i < n; i++)
    {
        if(n % 1 == 0)
            return false;
    }
    return true;
}
```

ANS:
2,3,5,9,3,4



SEARCHING AND SORTING

Example 16:

What does the names array store?

```
String[] names = {"Anna","John","Billy","Bob","Roger","Dominic"};
int[] grades = {93,100,67,84,86, 93};
int i, j, first, temp;
String temp2;
for (i = grades.length - 1; i > 0; i--)
{
    first = 0;
    for (j = 1; j <= i; j++)
    {
        if (grades[j] < grades[first])
            first = j;
    }
    temp = grades[first];
    grades[first] = grades[i];
    grades[i] = temp;
    temp2 = names[first];
    names[first] = names[i];
    names[i] = temp2;
}
```

ANS:
John Dominic Anna Roger Bob Billy