Day 16 - 09th July 2025

ID: mvsnarav

Name:Aravind kasanagottu

Selection Sort:

Task 👍1:

Write an algorithm / steps for selection sort.

1. Set min to position 0.
2. Repeat until the last position of the list:  
   a. Search for the smallest element in the list from min to the end.  
   b. Swap the element at min with the smallest element found.  
   c. Increment min by 1 to point to the next position.
3. End (the list is now sorted)

 Task 2:

Write a pseudo code for the selection sort

SelectionSort(A, n)

for i = 1 to n - 1 do

min\_j = i

min\_x = A[i]

for j = i + 1 to n do

if A[j] < min\_x then

min\_j = j

min\_x = A[j]

A[min\_j] = A[i]

A[i] = min\_x

Task 3:

Wap to make sure your list is sorted using selection sort.

public class SelectionSort {

public static void main(String[] args) {

int[] arr = {10, 20, 5, 46, 80};

int n = arr.length;

System.out.print("Before Sorting: ");

for (int i = 0; i < n; i++)

System.out.print(arr[i] + " ");

System.out.println();

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Swap arr[i] and arr[minIndex]

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

System.out.print("After Sorting: ");

for (int i = 0; i < n; i++)

System.out.print(arr[i] + " ");

}

}

Bubble Sort:

Task 4:

Write algorithm for the Bubble sort.

Start from the first element of the list.

Compare each pair of adjacent elements:

* If the left element is bigger than the right, swap them.

After one pass, the largest element moves to the end.

Repeat the process for the remaining elements (excluding the last sorted elements each time).

Continue until the entire list is sorted.

Task 5:

Write pseudo code for the bubble sort

BubbleSort(A, n)

for i = 0 to n - 2 do

for j = 0 to n - i - 2 do

if A[j] > A[j + 1] then

// Swap A[j] and A[j + 1]

temp = A[j]

A[j] = A[j + 1]

A[j + 1] = temp

Task 6:

Wap to make sure your list is sorted using Bubble sort.

public class BubbleSort {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 6};

int n = arr.length;

System.out.print("Before Sorting: ");

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

System.out.print(arr[i] + " ");

}

System.out.println();

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap arr[j] and arr[j + 1]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

System.out.print("After Sorting: ");

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

System.out.print(arr[i] + " ");

}

}

}

Insertion sort:

Task 7:

Write an algorithm for the Insertion sort.

Start from the second element in the list.  
Take the current element (key) and compare it with the elements to its left.  
 Shift all bigger elements one position to the right to make space.  
 Place the key in its correct position.  
 Move to the next element and repeat until the entire list is sorted.

Task 8:

Write pseudocode for the Insertion sort

InsertionSort(A, n)

for i = 1 to n - 1

key = A[i]

j = i - 1

while j >= 0 and A[j] > key

A[j + 1] = A[j]

j = j - 1

A[j + 1] = key

Task 9:

Wap to make sure your list is sorted using Insertion sort.

public class InsertionSort {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 6};

int n = arr.length;

System.out.print("Before Sorting: ");

for (int num : arr) {

System.out.print(num + " ");

}

System.out.println();

// Insertion Sort

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

System.out.print("After Sorting: ");

for (int num : arr) {

System.out.print(num + " ");

}

}

}

Task 10:

What are the advantages and disadvantages of Bubble sort Algo?

List them

**Advantages:**  
Very easy to understand and implement.  
 Works well for small or nearly sorted lists.

**Disadvantages:**  
Very slow for large lists.  
Takes many passes even if the list is almost sorted.

Poor performance - limitations of bubble sort

TAsk 11:

Algo for merge sort,

If the list has **1 element, it is already sorted.**  
 Otherwise**, divide the list into two halves.  
Recursively sort each half** using Merge Sort.  
**Merge the two sorted halves into a single sorted list..**

Task 12

pseudo code for merge sort,

MergeSort(A, left, right)

if left < right then

mid = (left + right) / 2

MergeSort(A, left, mid)

MergeSort(A, mid + 1, right)

Merge(A, left, mid, right)

Merge(A, left, mid, right)

create temp arrays L[] and R[]

copy data into L[] and R[]

merge L[] and R[] back into A[left..right] in sorted order

TSK 13

code for Merge sort

public class MergeSort {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 6};

int n = arr.length;

System.out.print("Before Sorting: ");

for (int num : arr)

System.out.print(num + " ");

System.out.println();

mergeSort(arr, 0, n - 1);

System.out.print("After Sorting: ");

for (int num : arr)

System.out.print(num + " ");

}

static void mergeSort(int[] arr, int left, int right) {

if (left < right) {

int mid = (left + right) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

static void merge(int[] arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int[] L = new int[n1];

int[] R = new int[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

}

Task 14:

Algo fro quick sort

Pick a **pivot element** from the list.  
**Partition the list** so:

* Elements **smaller than pivot go to the left.**
* Elements **greater than pivot go to the right.**

**Recursively apply Quick Sort** to the left part**.  
Recursively apply Quick Sort** to the right part.  
 The list is now sorted.

Task 15:

Pseudo code for quick sort

QuickSort(A, low, high)

if low < high then

p = Partition(A, low, high)

QuickSort(A, low, p - 1)

QuickSort(A, p + 1, high)

Partition(A, low, high)

pivot = A[high]

i = low - 1

for j = low to high - 1 do

if A[j] < pivot then

i = i + 1

swap A[i] and A[j]

swap A[i + 1] and A[high]

return i + 1

Task 16:

Code for Quick sort

public class QuickSort {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 6};

int n = arr.length;

System.out.print("Before Sorting: ");

for (int num : arr)

System.out.print(num + " ");

System.out.println();

quickSort(arr, 0, n - 1);

System.out.print("After Sorting: ");

for (int num : arr)

System.out.print(num + " ");

}

static void quickSort(int[] arr, int low, int high) {

if (low < high) {

int p = partition(arr, low, high);

quickSort(arr, low, p - 1);

quickSort(arr, p + 1, high);

}

}

static int partition(int[] arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

i++;

// Swap arr[i] and arr[j]

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// Swap arr[i + 1] and arr[high] (pivot)

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

}

Task 17:

Which one is better merge or bubble in terms of time complexity.

Merge Sort is better than Bubble Sort in terms of time complexity.

* **Merge Sort:** O(n log n) – much faster, especially for **large data and huge databases**.
* **Bubble Sort:** O(n^2) – very slow when the list is big.

In industries and huge databases, Merge Sort is preferred because it handles large datasets efficiently and sorts faster.

A comparison table (time, space, stability)  
 A small real-life example where Merge is faster than Bubble

Task 18:

Leet code qn:

Find the time complexity of the given merge operation between two sorted array.

Note: O(n + m) as we traverse both arrays once, n and m are size of both the arrays

<https://leetcode.com/problems/merge-sorted-array/description/>

Task 19:

This code is going overflow of stack.. Can you plz help me fix it guys.. ☹️

Plz be careful: Because recursive calls consume stack memory for every invocation and excessive depth can exceed system limits also..

public class RecLoop {

     public int calc(int n) {

        if (n == 0) return 0;

        return n + calc(n);

need to **reduce n in each recursive call** so it eventually reaches 0. For example:

**java**

**CopyEdit**

**return n + calc(n - 1);**

    }

Task 20:

public class BinarySearchNew {

    public int search(int[] arr, int toFind) {

        int left = 0, right = arr.len - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (arr[mid] == toFind) {

                return mid;

            } else if (arr[mid] < toFind) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return -1;

    }

}

 binary search function can we use in unsorted list?

Will it give you correct index no?

### **❓ Can we use Binary Search on an unsorted list?**

**No, we cannot use Binary Search on an unsorted list.**

Binary Search **works by comparing** toFind **with the middle element** and deciding whether **to** search **left** or **right** based on **sorted order.**

If the list is **not sorted,** you **cannot determine which half may contain the element, so binary search will fail**.

================================================================================================================================================

Add ons:

1.

What is the difference between binary tree and binary search tree (bst)

Can you explain diff between structure and operation of Binary tre and BST.

**Binary Tree:**

* A tree where each node has at most two children (left and right).
* There is **no specific order** in how the child nodes are arranged.
* Mainly used for **hierarchical data representation, expression trees, Huffman coding**, etc.

**Binary Search Tree (BST):**

* A **special type of binary tree** with an ordering property:
  + All nodes in the **left subtree are smaller** than the parent node.
  + All nodes in the **right subtree are larger** than the parent node.
* Used to **efficiently search, insert, and delete elements**.

**Structure Difference:**

* Binary Tree: No ordering rule, any value can be at any node.
* BST: Maintains order (Left < Root < Right).

**Operation Difference:**

* Binary Tree: Mainly used for **traversals** (inorder, preorder, postorder) and general tree operations.
* BST: Used for **fast searching, insertion, and deletion operations (O(log n) on average)**.

2.

In sorted array why do you think binary search tree is best than linear search.. Can you ecplain plz

**Linear Search:**

* Checks each element one by one from the start.
* Takes **O(n)** time in the worst case.
* Very slow for large arrays.

**Binary Search:**

* Works on **sorted arrays** only.
* Repeatedly divides the search space in half.
* Takes **O(log n)** time in the worst case.
* Very fast for large arrays.

**Example:**  
For an array of size 1,000,000:

* Linear Search may take up to 1,000,000 steps.
* Binary Search will take around 20 steps.

**Conclusion:**  
Binary Search is **best for searching in sorted arrays** as it is much faster and efficient compared to linear search.

3.

Difference between static and dynamic arrays.. Plz list them

**Static Array:**

* **Fixed size** decided at the time of creation.
* Uses int[] arr = new int[5];
* Size **cannot be changed** after creation.
* Memory is allocated at **compile-time**.
* Fast access, but **wastes space if unused**.

**Dynamic Array:**

* **Resizable size** that can grow/shrink during runtime.
* Uses **ArrayList<Integer> arr = new ArrayList<>();**
* Size can **change dynamically as elements are added or removed**.
* Memory is managed at **runtime**.
* Useful when you don't know the exact size in advance.

4.

In BFS, DFS which one is more preferred in terms of shortest path for the unweighted graphs.

Note:

 BFS explores nodes in increasing distance order from the source, ensuring shortest paths are found first.

**BFS (Breadth-First Search) is preferred for finding the shortest path in unweighted graphs.**

🔹 **Reason:**

* BFS **explores nodes level by level** (in increasing distance from the source).
* It ensures the **first time you reach a node, you are using the shortest path to reach it.**
* Guarantees **shortest paths in terms of the minimum number of edges.**

**DFS (Depth-First Search)**:

* Explores deep into one path before exploring others.
* Does **not guarantee the shortest path**, as it may reach a node via a longer path first.

**Conclusion:**  
For **shortest paths in unweighted graphs, always use BFS.**

5.

Write a code to implement a stack using an array.

Note: plz ensure bounds are checked to avoid overflow/underflow

6.

Home Task 👍

Pre order and post order traversal..

Recursion: assigned on Day 14th 02nd July 2025..

Wap to find the factorial of a number

Wap to find the Fibonacci series of a number

What is the difference between recursion and iteration

Wap to reverse a string using recursion..

Assigned on Day 14 - 03rd july 2025

Carry forward examples till 10 th july

1. Write a recursive function to search for an element in an array

2. Write a recursive function to count the digits of a positive integer (do also for sum of digits)

3. Write a recursive function to reverse a null-terminated string

4. Write a recursive function to convert a decimal number to binary

5. Write a recursive function to check if a string is a palindrome or not

6. Write a recursive function to copy one array to another