materio.

Scan to read online

September 25, 2025 • Paper Solution • 8 minute read

# DAA Mid Sem

*This resource covers all the questions with solutions from design and analysis of alogorithms mid semester exam.*

## Multiple Choice Questions

**In recursion, the base case represents:**

- A termination condition ✓
- A loop
- None of the mentioned
- An infinite call

**Which of the following is not a Divide and Conquer algorithm?**

- Linear Search ✓
- Merge Sort
- Strassen’s Matrix Multiplication
- Quick Sort

**Which of the following is not a property of an algorithm?**

- Ambiguity ✓
- Finiteness
- Input
- Definiteness

**The best case time complexity for Binary Search is:**

- $O(n \log n)$
- $O(n)$
- $O(1)$ ✓
- $O(\log n)$

**Which notation gives a tight bound on an algorithm's growth?**

- Big $\Theta$ ✓
- Big $O$
- Little $o$
- Big $\Omega$

**The worst case time complexity of Bubble Sort is:**

- $O(n)$
- $O(n \log n)$
- $O(\log n)$
- $O(n^2)$ ✓

**The Activity Selection Problem is optimally solved using:**

- Divide and Conquer
- Dynamic Programming
- Brute Force
- Greedy Strategy ✓

**Kruskal's algorithm is designed to find:**

- A Minimum Spanning Tree ✓
- A matrix product
- A shortest path
- A tree traversal

**Which algorithm uses a priority queue to compute shortest paths?**

- Dijkstra's ✓
- Kruskal's
- Bubble Sort
- Prim's

**The notation that provides an upper bound on the running time of an algorithm is called _____.**

**Answer:** O, Big O, Big Oh, Big oh ✓

**The algorithm that selects the minimum weight edge that does not form a cycle in a graph to build a Minimum Spanning Tree is _____**

**Answer:** Kruskal, kruskals, kruskal’s ✓

**In the Divide and Conquer technique, a problem is divided into smaller subproblems, solved recursively, and then the solutions are _____ to solve the original problem.**

**Answer: merged** ✓

**In algorithm analysis, the case where the input causes the algorithm to take the least amount of time is known as the _____ case.**

**Answer: Best** ✓

**The recurrence relation $T(n) = 2T(n/2) + O(n)$ corresponds to the time complexity of _______ sort using the divide and conquer method.**

**Answer: merge** ✓

---

## Programming Tasks

### Bubble Sort

You are given an unsorted array of integers. Your task is to implement a program to sort the array in **ascending order** using the **Bubble Sort algorithm**. Print the final sorted array in the specified output format.

**Input Format:**

- A single line of space-separated integers representing the elements of the array.

**Output Format:**

- Print the sorted array where to are sorted integers in ascending order, each separated by a single space.

**Constraints:**

- The array will contain **1 to 100** integers.
- The values of the integers will be in the range **[-10^4 to 10^4]**.

**File Name:** `Bubblesort.c`

```c
#include <stdio.h>

void bubbleSort(int arr[],int n){
    int i,j,temp;
    for(i=0;i<n;i++){
        for(j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
}

int main(){
    int n;
    scanf("%d",&n);

    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }

    bubbleSort(arr,n);

    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
}
```

## Min-Max Using Divide and Conquer

Implement a program to find **Min-Max in a given array** using the **Divide and Conquer** technique. The function should take three arguments: `(arr, low, high)` and recursively determine the minimum and maximum elements.

### Input Format:

- The first line contains an integer N, the number of elements in the array.
- The second line contains N space-separated integers representing the elements of the array.

### Output Format:

- Print the **minimum** and **maximum** element from the array using two separate lines in the following format:
    - Minimum:

  - Maximum:

**Constraints:**

- $2 \leq N \leq 100$
- $-1000 \leq arr[i] \leq 1000$

**File Name:** `MinMax.c`

```c
#include <stdio.h>

void findMinMax(int arr[],int low,int high,int *min,int *max){
    if(low==high){
        *min=arr[low];
        *max=arr[low];
        return;
    }
    int mid=(low+high)/2;
    int leftMin,leftMax,rightMin,rightMax;

    findMinMax(arr,low,mid,&leftMin,&leftMax);
    findMinMax(arr,mid+1,high,&rightMin,&rightMax);

    *min=(leftMin<rightMin)?leftMin:rightMin;
    *max=(leftMax>rightMax)?leftMax:rightMax;
}

int main(){
    int n;
    scanf("%d",&n);

    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }

    int minVal,maxVal;
    findMinMax(arr,0,n-1,&minVal,&maxVal);

    printf("Minimum: %d\\n",minVal);
    printf("Maximum: %d\\n",maxVal);
}
```

---

## Search Insert Position

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

**Input Format:**

- The first line of input contains an integer N, representing the number of elements in the array.
- The second line contains N space-separated integers sorted in **increasing order**.
- The third line contains an integer representing the **target** value.

**Output Format:**

- Print a single integer — the index at which the target is found, or the index where it should be inserted.

**Constraints:**

- $$1 \leq N \leq 100$$
- $-10^4 \leq \text{Array elements, Target} \leq 10^4$
- All array elements are distinct and sorted in increasing order.

**File Name:** `Search.c`

```c
#include <stdio.h>
int searchInsert(int* nums, int numsSize, int target) {
    int left = 0, right = numsSize - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right  = mid - 1;
        }
    }
    return left;
}

int main() {
    int N;
    scanf("%d", &N);

    int nums[N];
    for(int i = 0; i< N; i++) {
        scanf("%d", &nums[i]);
    }
    int target;
    scanf("%d", &target);

    int index = searchInsert(nums, N, target);
    printf("%d\\n", index);

    return 0;
}
```

## Mahishmati Sword Search

In the majestic kingdom of **Mahishmati**, the royal arsenal contains a **sorted array** of swords based on their strength. When **Baahubali** is about to go to war, he seeks a mightier sword than what he currently holds. But time is short — he needs to find this in the fastest way possible.

**Problem Statement:** Given a sorted array of integers representing the strength of swords in ascending order (duplicates allowed), and an integer x representing the strength Baahubali currently holds, find the **smallest sword strength strictly greater than x**. If no such sword exists, return -1.

**Input Format:**

- First line: Two integers n and x n: Number of swords x: Baahubali's current strength

- Second line: n space-separated integers denoting the strengths of the swords (sorted in non-decreasing order)

**Output Format:**

- A single integer — the strength of the **smallest sword strictly greater than x**, or -1 if no such sword exists.

**Constraints:**

- $$1 \leq n \leq 10^5$$
- $$-10^9 \leq arr[i], x \leq 10^9$$
- arr is sorted in non-decreasing order

**Explanation of Test Case:**

- **Input:** 5 10 5 10 10 20 30
- **Output:** 20
- **Explanation:** Baahubali's current strength is 10. Among the swords, the next stronger sword is 20. Hence, the output is 20.

**File Name:** `Mahishmati.c`

```c
#include <stdio.h>

int find(int n,int x, int arr[]) {
    int left =0, right= n-1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] <= x) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return (left < n) ? arr[left] : -1;
}

int main() {
    int n, x;
    scanf("%d %d", &n, &x);
    int arr[n];
    for (int i= 0; i<n; i++) {
        scanf("%d", &arr[i]);
    }
    int result = find(n, x, arr);
    printf("%d\\n", result);
    return 0;
}
```

---

### Operation SINDOOR: Fractional Knapsack

During **Operation SINDOOR**, a strategic tri-services mission post the Pahalgam terror attack, a critical logistics challenge emerged. Aerial units were tasked with dropping high-value military supplies to border units under constant threat of UAVs and drone strikes. Each supply item had an **impact value** based on intelligence reports and a **weight** determining drone load. You must help the Integrated Command and Control System (ICCS) **maximize mission impact** with limited drone carrying capacity.

**Problem Statement:** Given n supply packages, each with a value $v_i$ and weight $w_i$, and a total drone capacity W, you can **fractionally divide** supplies to load the drone. Return the **maximum possible total value** (to 2 decimal places) of supplies that can be delivered during the supply drop mission.

**Input Format:**

- An integer n (number of supply items)
- A float W (maximum drone weight capacity)
- n lines, each containing two space-separated integers $v_i$ and $w_i$

**Output Format:**

- A single float: maximum value the drone can carry (rounded to 2 decimal places)

**Constraints:**

- $$1 \leq n \leq 10^5$$
- $$1 \leq W \leq 10^4$$
- $$1 \leq v_i, w_i < 10^4$$
- Output must be accurate to **2 decimal places.**

**Explanation of Test Case:**

- **Input:** 3 50 60 10 100 20 120 30
- **Output:** 240.00
- **Explanation:**
  - Drone carries full package 1 → 10kg for 60 impact
  - Drone carries full package 2 → 20kg for 100 impact
  - Drone carries 20kg (2/3) of package 3 → (2/3 × 120) = 80 impact
  - **Total impact value** = **60** + **100** + **80** = **240.00**

**File Name:** `Sindoor.c`

C

```c
#include <stdio.h>
#include <stdlib.h>

// Structure to store supply items
struct Item {
    int value, weight;
    double ratio;
};

// Comparator function for sorting by value/weight ratio (descending)
int cmp(const void *a, const void *b) {
    double r1 = ((struct Item*)a)->ratio;
    double r2 = ((struct Item*)b)->ratio;
    if (r1 < r2) return 1;
    else if (r1 > r2) return -1;
    return 0;
}

int main() {
    int n;
    double W;
    scanf("%d", &n);
    scanf("%lf", &W);

    struct Item items[n];

    for (int i = 0; i < n; i++) {
        scanf("%d %d", &items[i].value, &items[i].weight);
        items[i].ratio = (double)items[i].value / items[i].weight;
    }

    // Sort items by value-to-weight ratio in descending order
    qsort(items, n, sizeof(struct Item), cmp);

    double maxValue = 0.0;

    for (int i = 0; i < n && W > 0; i++) {
        if (items[i].weight <= W) {
            maxValue += items[i].value;
            W -= items[i].weight;
        } else {
            maxValue += items[i].ratio * W;
            W = 0;
        }
    }

    printf("%.2lf\\n", maxValue);
    return 0;
}
```

materio.

*The InsightRoom*