**Semester: V**
**Subject Name: DAA & Code: CS2012**
**Branch (s): CSE, IT, CSCE**

# AUTUMN MAKE UP MID SEMESTER EXAMINATION-2023

School of Computer Engineering
Kalinga Institute of Industrial Technology, Deemed to be University
Design and Analysis of Algorithms
[CS2012]

**Time: 1 1/2 Hours** **Full Mark: 40**

***Answer All the Questions.***
***The figures in the margin indicate full marks. Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.***

1. Answer all the questions. [ 2 x 5 = 10 ]

a)

I. What is the time complexity of the following function, fun()?

```
int fun(int n){
    int i, j, s=0;
    for(i= 0; i<n; i= i+1)
       for(j = 0; j*j < n; j= j + 1)
              s=s+j;
    return s;
}
```

A. O(n) B. O(n*log(n)) C. O(n^2) D. O(n√n) E. NONE

II. What exact value will the function fun() return if the value of 'n' is 4?

b) Suppose that an algorithm takes eighty seconds to run on an input size (n) of 120. Estimate the instances (input size) that can be processed in 560 seconds. Assume that the algorithm complexity is θ (n).

c)

I. Show that $2^x = O(3^x)$.

II. Show that $n! = O(n^n)$.

d) Solve the following recurrence by the master theorem.
$T(n) = \sqrt{2}T(n/2) + \log n$, $T(1)=1$

e) Consider the weights and values of the items listed below. Note that there is only one unit for each item.

| Item No. | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Weight (Kg) | 10 | 7 | 4 | 2 |
| Value (Rs.) | 60 | 28 | 20 | 24 |

The task is to pick a subset of these items such that their total weight is no more than 11 Kg, and their total value is maximized. You can't split the item. The total value of items picked by an optimal algorithm is denoted by $V_{opt}$. The total value of items picked by the greedy algorithm is denoted by $V_{greedy}$. Describe and find the value of $V_{opt} - V_{greedy}$.

2. [ 5 x 2 =10 ]

a) Write the MERGE-SORT (A, p, r, order) procedure where, at each step, it divides the array or sub-array into two parts such that the first part contains elements twice of second part instead of dividing in the middle. A is the array to be sorted, order is either "asc" for ascending or "desc" for descending. p and r are the starting and ending indexes of the sub-array, respectively.

b) Describe in a step-by-step process how the above algorithm is applied on the following list to sort the data in descending order.

50, 40, 30, 60, 20, 50, 90, 80, 70, 30, 60, 10

3. [ 10 Marks ]

Let *A* be a heap of size *n*. Write the most efficient algorithm/program for the following tasks and find the time complexity of each task.

- Find the sum of all even numbers in *A.*
- Find the sum of the largest $\boldsymbol{log_2 n}$ elements in *A.*

4. [ 10 Marks ]

Alice stores the message **KIITKIITEE2023200003** in a file named kiit.txt. Her aim is to minimize the size of the file. Design an efficient algorithm for compressing the file size and solve the following:

- Find the optimal code (prefix code) of each character.
- Find the average code length.
- Calculate the length of the encoded message for KIIT2023 (in bits).

*** Best of Luck ***

*SOLUTION & EVALUATION SCHEME*

1. Answer all the questions. [ 2 x 5 = 10 ]

a)

I. What is the time complexity of the following function, fun()?

```
int fun(int n){
    int i, j, s=0;
    for(i= 0; i<n; i= i+1)
       for(j = 0; j*j < n; j= j + 1)
              s=s+j;
    return s;
}
```

B. O(n) B. O(n*log(n)) C. O(n^2) D. O(n√n) E. NONE

II. What exact value will the function fun() return if the value of 'n' is 4?

**Scheme:**

Correct answer: 2 Marks

Wrong answer, but explanation approaches to answer: Step Marking

**Answer:** I. O(n√n) (correct answer 1 mark, wrong answer 0 mark)

II. 4 (correct answer 1 mark, wrong answer 0 mark)

**Explanation:** The outer loop runs from i = 0 to i < n, which means it executes n times. The inner loop runs from j = 0 to j * j < n, which means it executes sqrt(n) times for each value of i. Therefore, the total number of times the inner loop executes is n * sqrt(n). The statement s = s + j inside the inner loop executes the same number of times as the inner loop. Therefore, the time complexity of the code is O(n * sqrt(n)). This means that the running time of the code grows proportionally to n * sqrt(n) as the input size n increases.

b) Suppose that an algorithm takes eighty seconds to run on an input size (n) of 120. Estimate the instances (input size) that can be processed in 560 seconds. Assume that the algorithm complexity is θ (n).

**Scheme:**

Correct answer: 2 Marks

Wrong answer, but explanation approaches to answer: Step Marking

As f(n) is in θ (n)

c*120 = 80 sec

⇒ c = 80/120 = 2/3

Now need to calculate **n** for t = 560

⇒ c n= 560

⇒ n = 560 x 3/2

⇒ n = 840

Hence, the maximum input possible is 840.

c)

I. Show that $2^x = O(3^x)$.
II. Show that $n! = O(n^n)$.

**Scheme:**

Correct answer: 2 Marks

Wrong answer, but explanation approaches to answer: Step Marking

I. Show that $2^x = O(3^x)$

Proof 1:

This is easy to see since $\lim_{x \to \infty} \frac{2^x}{3^x} = 0$

Proof 2:

If x>=1, then clearly $(3/2)^x \geq 1, so$

$$2^x \leq 2^x \left(\frac{3}{2}\right)^x = \left(\frac{2\ x\ 3}{2}\right)^x = 3^x$$

II. Show that $n! = O(n^n)$

Proof:

Notice that when n>=1, 0<=n! = 1. 2. 3. ….n <= n.n.n…..n = $n^n$

Therefore, $n! = O(n^n)$ (Here $n_0$=1 and c = 1)

d) Solve the following recurrence by the master theorem.
T(n) = √2T(n/2) + log n, T (1)=1

**Scheme:**

Correct answer: 2 Marks

Wrong answer, but explanation approaches to answer: Step Marking

T(n) = aT(n/b) + f(n)

In this case:

- a = √2
- b = 2
- f(n) = log n

Now, let's calculate $n^{\log_b a}$:

n^log_2(√2) = n^0.5

Since this is in Case 1, T(n) = Θ ($n^{\log_b a}$) = **Θ (n^0.5)**

e) Consider the weights and values of the items listed below. Note that there is only one unit for each item.

| Item No. | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Weight (Kg) | 10 | 7 | 4 | 2 |
| Value (Rs.) | 60 | 28 | 20 | 24 |

The task is to pick a subset of these items such that their total weight is no more than 11 Kg, and their total value is maximized. You can't split the item. The total value of items picked by an optimal algorithm is denoted by $V_{opt}$. The total value of items picked by the greedy algorithm is denoted by $V_{greedy}$. Describe and find the value of $V_{opt} - V_{greedy}$.

**Scheme:**

Correct answer: 2 Marks

Wrong answer, but explanation approaches to answer: Step Marking

**Answer:**

Firstly, we can pick item_4 (Value weight ratio is highest). Second highest is item_1, but cannot be picked because of its weight. Now item_3 shall be picked. item_2 cannot be included because of its weight. Therefore, overall profit by Vgreedy = 20+24 = 44. Hence, **Vopt – Vgreedy = 60-44 = 16.** So, answer is **16**.

2. [ 5 x 2 =10 ]

a) Write the MERGE-SORT (A, p, r, order) procedure where, at each step, it divides the array or sub-array into two parts such that the first part contains elements twice of second part instead of dividing in the middle. A is the array to be sorted, order is either "asc" for ascending or "desc" for descending. p and r are the starting and ending indexes of the sub-array, respectively.

**Scheme:**

Correct answer: 5 Marks

Wrong answer, but explanation approaches to answer: Step Marking

```
// A function that implements merge sort on A[] using a modified division
strategy
void mergeSort(int A[], int p, int r, char* order)
{
    if (p < r) {
        // Find the point to divide the array into two parts such that the
first part contains elements twice of second part

          int q = (2*r+p-1)/3;

          // Sort the first and second parts recursively
        mergeSort(A, p, q, order);
        mergeSort(A, q + 1, r, order);

        // Merge the sorted parts
        merge(A, p, q, r, order);
    }
}

// A function to merge two subarrays of A[]
void merge(int A[], int p, int q, int r, char* order)
{
  // Find the sizes of the two subarrays to be merged
    int n1 = q - p + 1;
    int n2 = r - q;
  int i, j;
    // Create temp arrays
    int L[n1], R[n2];

    // Copy data to temp arrays L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = A[p + i];
    for (j = 0; j < n2; j++)
        R[j] = A[q + 1 + j];

    // Merge the temp arrays back into A[p..r]
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    int k = p; // Initial index of merged subarray
```

```
if (strcmp(order, "asc") == 0) { // If order is ascending
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                A[k] = L[i];
                i++;
            }
            else {
                A[k] = R[j];
                j++;
            }
            k++;
        }
    }
    else if (strcmp(order, "desc") == 0) { // If order is descending
        while (i < n1 && j < n2) {
            if (L[i] >= R[j]) {
                A[k] = L[i];
                i++;
            }
            else {
                A[k] = R[j];
                j++;
            }
            k++;
        }
    }

    // Copy the remaining elements of L[], if there are any
    while (i < n1) {
        A[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[], if there are any
    while (j < n2) {
        A[k] = R[j];
        j++;
        k++;
    }
}
```

b) Describe in a step-by-step process how the above algorithm is applied on the following list to sort the data in descending order.

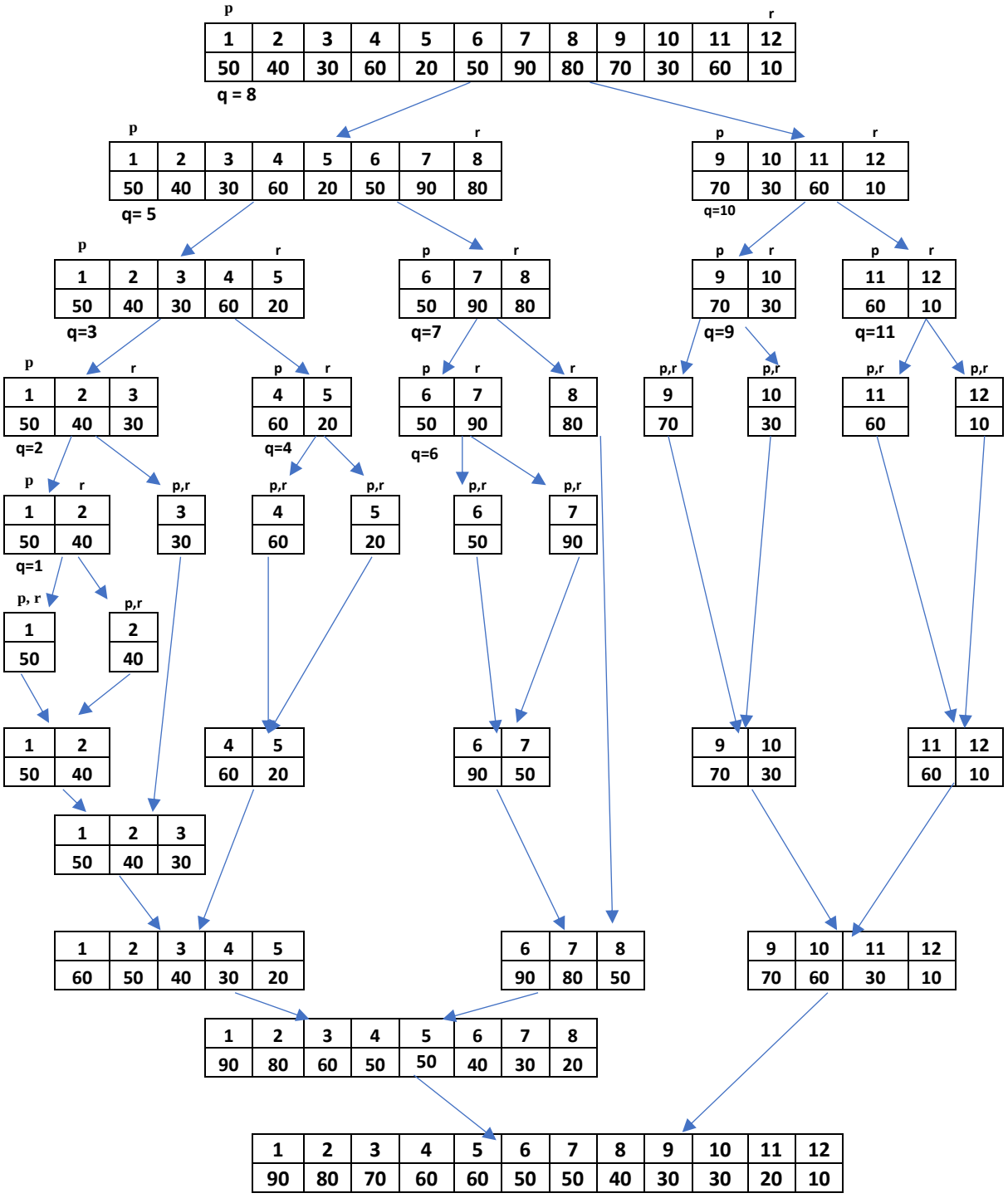
50, 40, 30, 60, 20, 50, 90, 80, 70, 30, 60, 10

**Scheme:**

Correct answer: 5 Marks

Wrong answer, but explanation approaches to answer: Step Marking

```
q = (2*r+p-1)/3
```



3. [ 10 Marks ]

Let *A* be a heap of size *n*. Write the most efficient algorithm/program for the following tasks and find the time complexity of each task.

- Find the sum of all even numbers in *A*.
- Find the sum of the largest $\boldsymbol{log_2 n}$ elements in *A*.

**Scheme:**

Find the sum of all even numbers:
Algorithm or program: 3 Marks, Complexity: 1 Mark
Find the sum of the largest $\boldsymbol{log_2 n}$ elements in *A:*
Algorithm or program: 5 Marks, Complexity: 1 Mark
Wrong answer, but explanation approaches to answer: Step Marking

- To find the sum of all even numbers in A, we can use the following algorithm:

```
// A function to find the sum of all even numbers in a heap A of size n
int sumEven(int A[], int n)
{
    // Initialize the sum to zero
    int sum = 0;

    // Loop through all the elements of the heap
    for (int i = 0; i < n; i++)
    {
        // If the element is even, add it to the sum
        if (A[i] % 2 == 0)
        {
            sum += A[i];
        }
    }

    // Return the sum
    return sum;
}
```

The time complexity of this algorithm is **O(n)**, since we have to iterate through all the elements of the heap once.

- To find the sum of the largest $\log_2 n$ elements in A, we can use the following algorithm:

```
// Function to maintain the max-heap property
void maxHeapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        // Swap arr[i] and arr[largest]
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        // Recursively heapify the affected sub-tree
        maxHeapify(arr, n, largest);
    }
}

// Function to build a max-heap
void buildMaxHeap(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        maxHeapify(arr, n, i);
    }
}

// Function to find the sum of the largest lgn elements in the heap
int sumOfLargestLgnElements(int arr[], int n) {
    int sum = 0;
    int lgn = (int)(log2(n) + 1); // Calculate lgn

    // Extract and sum the largest lgn elements
    for (int i = 0; i < lgn; i++) {
        sum += arr[0]; // The maximum element is always at the root
        arr[0] = arr[n - 1]; // Replace the root with the last element
        n--; // Decrease the size of the heap
        maxHeapify(arr, n, 0); // Heapify the modified heap
    }

    return sum;
}
```

The time complexity of the above code is **O(n)**. Here is how, It can be calculated:

- The main function calls the buildMaxHeap function, which takes **O(n)** time to convert the array into a max-heap.
- The main function also calls the sumOfLargestLgnElements function, which takes **O(lgn * lg n)** time to extract and sum the largest lgn elements from the heap. This is because:
  - The function calculates lgn in **O(1)** time using the log2 function.
  - The function loops lgn times, and in each iteration, it does the following operations:
    - It adds the root element to the sum in **O(1)** time.
    - It replaces the root element with the last element in **O(1)** time.
    - It decreases the size of the heap by one in **O(1)** time.
    - It calls the maxHeapify function, which takes **O(log n)** time to restore the heap property.
- Therefore, the total time complexity of the code is **O(n + lgn * lg n)**, which is asymptotically equivalent to **O(n)**.

**Note#**

- Full marks can be awarded if students assume the array is in max-heap and the complexity of the above operation derived as O(lgn * lgn).
- Students may also approach this bit after sorting the heap in descending order.

```
//function to find the sum of the largest log2n elements in an array A, where
elements are stored in descending order.
int sumLargest(int A[], int n)
{
    // Initialize the sum to zero
    int sum = 0;

    // Find the number of elements to be added, which is log2n
    int k = (int) log2(n);

    // Loop through the k largest elements of the heap, which are stored in the
first k nodes
    for (int i = 0; i < k; i++)
    {
        // Add the element to the sum
        sum += A[i];
    }

    // Return the sum
    return sum;
}
```

The time complexity of the above function is **O(log n)** , since we only have to iterate through the first $\log_2 n$ elements of the array, which are arranged in decreasing order.

Time complexity of this procedure including heapsort is O **(nlog n)** + **O (log n) = O(n log n)**

4. [ 10 Marks ]

Alice stores the message **KIITKIITEE2023200003** in a file named kiit.txt. Her aim is to minimize the size of the file. Design an efficient algorithm for compressing the file size and solve the following:

- Find the optimal code (prefix code) of each character.
- Find the average code length.
- Calculate the length of the encoded message for KIIT2023 (in bits).

**Scheme:**

Algorithm or program: 2 Marks
Huffman Tree*: 3 Marks*
Prefix code: 2 Marks
Average code length: 2 Marks
Length of encoded message: 1 Mark
Wrong answer, but explanation approaches to answer: Step Marking

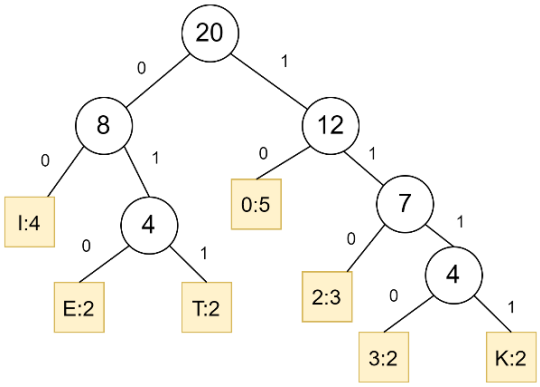
**Algorithm Huffman(C)**

```
1. n = |C|
2. Q = C                          // Q is a Min-Priority Queue.
3. for i=1 to n-1
4.       allocate a new node Z
5.       Z.left = x = EXTRACT-MIN(Q)
6.       Z.right = y = EXTRACT-MIN(Q)
7.       Z.freq = x.freq + y.freq
8.       INSERT(Q, Z)
9. return EXTRACT-MIN(Q)          // Return the root of the tree
```

- To compress the file size using Huffman coding, we need to follow these steps:
  - Calculate the frequency of each character in the file. For example, the frequency of I is 4, the frequency of 0 is 5, and so on.

| Character | I | 0 | 2 | T | E | K | 3 |
|---|---|---|---|---|---|---|---|
| Frequency | 4 | 5 | 3 | 2 | 2 | 2 | 2 |

  - Create a Huffman tree using the frequencies of the characters. The least frequent characters are placed at the bottom of the tree, and the most frequent ones are placed near the root. The internal nodes of the tree have the sum of the frequencies of their children as their values. One possible Huffman tree is as follows




***Note# Students may try with other tree structures. However, root value will be always 20.***

- Assign codes to each character by traversing the tree from the root to the leaf. The left edge is assigned 0 and the right edge is assigned 1. For example, the code for I is 00, the code for K is 1111, and so on.

- To find the optimal code (prefix code) of each character, we can use the codes that we assigned in above step. The optimal code for each character is shown in the table below:

| Character. | Frequency. | Code |
|---|---|---|
| I | 4 | 00 |
| 0 | 5 | 10 |
| 2 | 3 | 110 |
| T | 2 | 011 |
| E | 2 | 010 |
| K | 2 | 1111 |
| 3 | 2 | 1110 |

- To find the average code length, we can use the formula:

$$L_{avg} = \frac{\sum_{c \in C} c.freq * d_T(c)}{\sum_{c \in C} c.freq}$$

$d_T(c)$ = depth of char $c$ in the tree T

$c.freq$ = frequency of char $c$

For example, freq. of K=2 and depth or length=4. Plugging in the values from the table above, we get:

Average code length= (4×2+5×2+3×3+2×3+2×3+2×4+2×4)/20

Average code length=55/20

Average code length=**2.75**

- To calculate the length of the encoded message for KIIT2023 (in bits), we can use the codes of each character and count the number of bits.

For example, KIIT2023 is encoded as:

1111 (K) + 00 (I) + 00 (I) + 011 (T) + 110 (2) + 10 (0) + 10 (0) + 1110 (3)

The length of this encoded message is **22 bits**.