

Q1) Solve the following recurrence and find out asymptotic time complexity. Show complete working. [5 Marks]

$$T(n) = 5T(n/3) + \Theta(1)$$

$$a = 5$$

$$b = 3$$

$$c = 0$$

$$\frac{a}{b^c} = \frac{5}{3^0} = 5 > 1$$

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_3 5}\right)$$

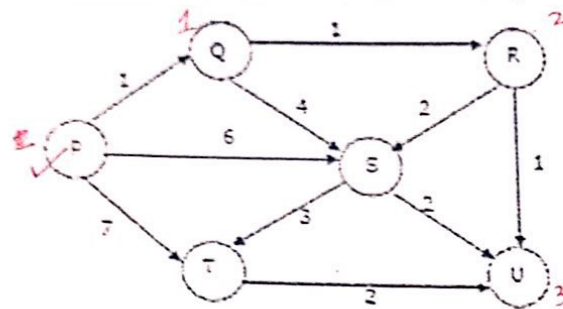
Q2) What is the worst case time complexity of the Quick sort? [1 Mark]

- a) $O(n \log n)$
b) $O(n)$

c) $O(n^3)$

☒ d) $O(n^2)$

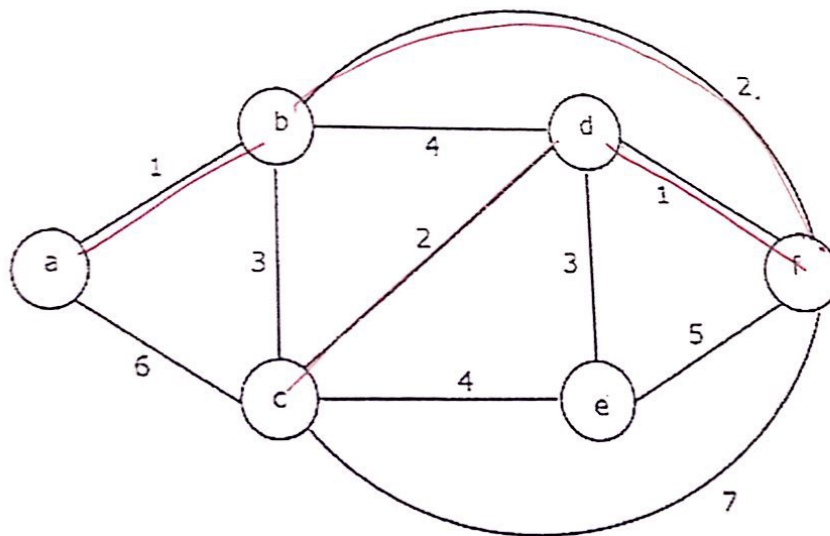
Q3) Suppose we run Dijkstra's single source shortest-path algorithm on the following directed graph with vertex P as the source. In what order do the nodes get included in the shortest path distances are finalized? [2 Marks]



PQRUST

- a) PQT SRU
b) PQT SUR
c) PQRSUT
☒ d) PQRUST
e) PSQRUT

Q4) Consider the following graph: [2 Marks]



Which one of the following cannot be the sequence of edges added, in that order, using Kruskal's algorithm?

- I. (a-b), (d-f), (b-f), (d-c), (d-e)
II. (a-b), (d-f), (d-c), (b-f), (d-e)
III. (d-f), (a-b), (d-c), (b-f), (d-e)
☒ IV. (d-f), (a-b), (b-f), (d-e), (d-c)

FAST School of Computing

Q5) When does the worst case of Quick sort occur? [1 Mark]

IF array is sorted

Q6) Show how to sort n integers in the range 0 to $n^3 - 1$ in $O(n)$ time. You can name the algorithm and explain why it will be linear. You do not need to write pseudocode of the algorithm but provide a justification why and how it will be linear. [4 Marks]

convert to base n

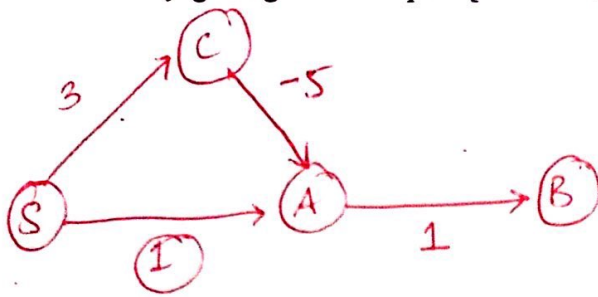
Use Radix Sort, with counting sort as intermediate sorting algorithm

Q7) Describe an algorithm that, given n integers in the range 0 to k , preprocesses its input and then answers any query about how many of the n integers fall into a range $[a \dots b]$ in $O(1)$ time. Your algorithm should use $\theta(n + k)$ preprocessing time. [3 Marks]

Calculate cumulative Array [from counting sort \rightarrow C array]

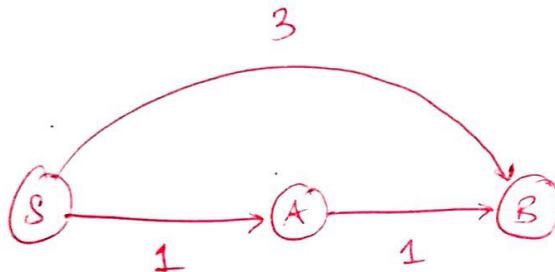
$$\text{Answer} = C[b] - C[a-1]$$

Q8) Dijkstra's algorithm does not guarantee shortest paths when the graph has negative edge weights. Justify this statement by giving an example. [2 Marks]



Q9) Adding a number w on the weight of every edge of a graph might change the shortest path between two vertices u and v . Is this statement True or False? Justify your answer with an example. [2 Marks]

YES

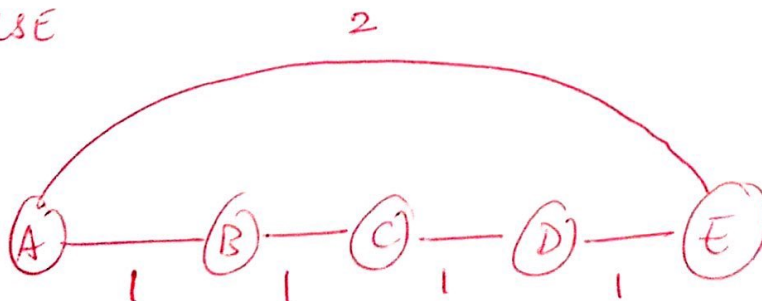


Shortest path S to B = SAB

Add 2 Now Shortest path is direct path

Q10) Let T be a minimum spanning tree of a graph G . Then for any two vertices u, v the path from u to v in T is a shortest path from u to v in G . Is this statement True or False? Justify your answer with an example. [2 Marks]

FALSE



Shortest path A to E is not in T

Q11) Given a weighted directed graph D , let T be its shortest path tree from a given vertex s . Let H be D with an extra edge uv . H has one extra edge from vertex u to vertex v and the weight of this edge is w . Design an efficient algorithm to find shortest path tree of H from the same vertex s . Analyze its time complexity. [4 Marks]

Apply Relax Function

$RELAX(u, v, w)$

$O(1)$ time

Some More Steps?

Q12) Given a weighted undirected graph G , let T be its minimum spanning tree. Let $H = G + uv$. H has an extra edge between vertices u and v and the weight of this edge is w . Design an efficient algorithm to find minimum spanning tree of H . Analyze its time complexity. [10 Marks]

Add uv to T , this will create one cycle

Find the cycle and remove heaviest edge

$O(n)$

Q13) Assume you are an awesome parent and want to give your children some cookies. But, you give each child at most one cookie. Each child i has a greed factor g_i , which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s_j . If $s_j \geq g_i$, we can assign the cookie j to the child i , and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number. Analyze its time complexity. [10 Marks]

Note:

You may assume the greed factor is always positive.

You cannot assign more than one cookie to one child.

Example 1:

Input: [1,2,3], [1,1]

Output: 1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3. And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content. You need to output 1.

Example 2:

Input: [3,4], [1,2,4,5]

Output: 2

Explanation: You have 2 children and 4 cookies. The greed factors of 2 children are 3, 4. You have 2 cookies whose sizes are big enough to gratify two children, You need to output 2.

```

Sort cookies & S      [Total m cookies]
Sort greed factor G   [Total n child]

output = 0
i = 1 (for child)
j = 1 (for cookie)

For While ( i ≤ n AND j ≤ m )
    IF ( S[j] ≥ G[i] )
        i++
        j++
        output ++
    ELSE
        j++

```

14) The longest increasing subsequence problem is to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous or unique. [10 Marks]

Please note that the problem specifically targets subsequences that need not be contiguous, i.e., subsequences are not required to occupy consecutive positions within the original sequences.

For example, the longest increasing subsequence of [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15] is [0, 2, 6, 9, 11, 15].

Following is a brute force recursive solution with exponential time complexity. For each item, there are two possibilities:

1. Include the current item in LIS if it is greater than the previous element in LIS and recur for the remaining items.
2. Exclude the current item from LIS and recur for the remaining items.

```
1. int LIS(int arr[], int i, int n, int prev)
2. {
3.     // Base case: nothing is remaining
4.     if (i == n) {
5.         return 0;
6.     }
7.     // case 1: exclude the current element and process the
8.     // remaining elements
9.     int excl = LIS(arr, i + 1, n, prev);
10.
11.    // case 2: include the current element if it is greater
12.    // than the previous element in LIS
13.    int incl = 0;
14.    if (arr[i] > prev) {
15.        incl = 1 + LIS(arr, i + 1, n, arr[i]);
16.    }
17.    // return the maximum of the above two choices
18.    return max(incl, excl);
19. }
20. int main()
21. {
22.     int arr[] = { 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15 };
23.     int n = sizeof(arr) / sizeof(arr[0]);
24.
25.     cout << "The length of the LIS is " << LIS(arr, 0, n, INT_MIN);
26.
27.     return 0;
28. }
```

Recursive calls



variables

2 variables so 2-D Array