DS2030: DSA for DS
End-Semester Exam - 180 minutes

Name: CK

Roll Number:

Please read the following instructions carefully

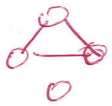
- This is a closed book, closed notes exam. Calculators are allowed. However laptops or mobile phones are **not** allowed.
- Use the space provided after every question for writing your answer. No Additional Sheets will be given for rough work. There are extra pages provided in the booklet for rough work.
- *You are **not** allowed to use Python list operations and other Python data structures/operations unless explicitly stated.*
- Be precise and concise in your answers.
- Include explanations, derivations, and examples when appropriate. This can fetch you partial scores even if the final answer is incorrect.
- Please write legibly.
- Work efficiently. Some questions are easier than others.
- **Good Luck.**

Section #	Max. Score	Score
Two Pointers	26	
Five Pointers	20	
Nine Pointers	14	
Total	60	

1 Two Pointers (2 x 13 = 26 points)

A penalty of -1 point for incorrect answers applies only to the True/False questions (no explanations required for those). Explanations are required only for Questions 1.10 to 1.14

1.1 A simple undirected graph with n vertices and m edges with $m = n-1$ is connected. True or False.

False 

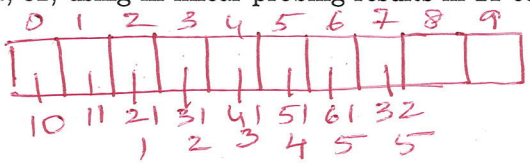
1.2 A digit-sorting procedure for creating the ascending order of numbers is called stable if it always places larger digits before smaller digits whenever two numbers have the same digit value in the current position. True or False.

False

1.3 In an AVL tree, inserting a node can cause height imbalance at any ancestor along the search path from the root to the inserted node—not necessarily the node where the insertion occurred. No other nodes can become unbalanced. True or False.

True

1.4 Given a hash table of size 10 and the hash function $h(k) = k \mod 10$, inserting 10, 11, 21,31, 41,51, 61, 32; using in linear probing results in 21 collisions. True or False.



False

20 collisions

1.5 Let (u, v) be an ordered edge labeled as back edge during directed depth first search traversal on a directed graph G. If $arr(x)$ and $dep(x)$ represent the arrival and departure times at a node x, then the back edge (u, v) satisfies $arr(u) < arr(v) < dep(v) < dep(u)$. True or False.

False $arr(v) < arr(u) < dep(u) < dep(v)$

1.6 Let T be a minimum spanning tree of a weighted graph G and e be an edge of G that is not in T. If C is the cycle formed by adding e to T. Then for every edge f in C, edge weight of $f \leq$ edge weight of e. True or False.

True

1.7 Inserting and removing a vertex in a graph containing n vertices that is stored as an adjacency matrix is of $O(n)$ - True or False.

False $O(n^2)$

1.8 A graph is bi-partite if the Breadth first traversal of the graph from any node does not result in a cross edge. True or False.

True

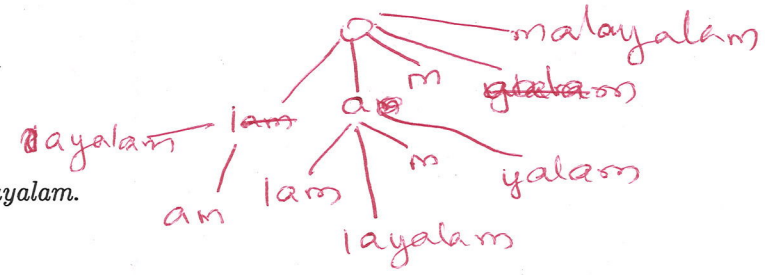
1.9 In a 2-4 tree, when a node overflows by acquiring a fifth child during insertion, the tree is re-structured by splitting the overflowing node and promoting the middle key to the parent. The overflowing stops at the parent node and does not extend up to the root of the tree. True or False

False

1.10 Prove that $n^2+10n+5$ is $O(n^2)$. Give the constants c and n_0 that would make the statement true.

$n_0 = 1 \quad C = 16.$

1.11 Construct the suffix trie for the pattern *malayalam*.



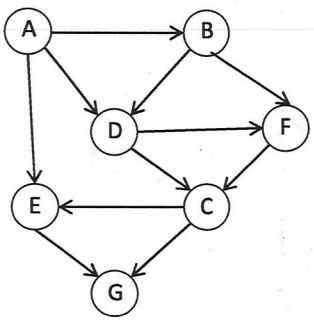
1.12 In a standard top-down mergesort, what is the number of merges at level i?

2^i

1.13 How does the Bellman-Ford algorithm detect a negative cycle in a weighted graph?

If the algorithm is executed once more after the first pass, and the cost at a node reduces, then there exists a negative cycle in the graph.

1.14 Consider the directed graph shown in Figure 1. Which of the following ordering of vertices is(are) a topological ordering?



A B D F C E G

Figure 1: Directed graph for question 1.14

(a) A, B, E, D, F, C, G
(b) A, B, C, D, F, E, G
(c) A, B, D, F, C, E, G
(d) A, E, B, D, F, C, G

2 Five pointers (4 x 5 = 20 points)

2.1 Given an array A containing N *distinct* elements in a random order, write the algorithm for conducting **inplace Quicksort**, where the *pivotal element is always chosen as the last element of the (sub)array.* You are to use only the array data structure (cannot use Python List data structure)

Refer slides.

2.2 Let all the edges of a weighted graph have distinct weights. Prove that Kruskal's principle returns the minimum spanning tree.

Let $g_1 < g_2 < g_3 < \cdots g_i < \cdots g_N$ be the sorted edges that are part of the MST returned by kruskal's principle. and let $f_1 < f_2 < f_3 < \cdots f_i \cdots < f_N$ be the true MST. let i be the first location of mismatch between both the trees. i.e., $g_1 = f_1, g_2 = f_2 \cdots g_{i-1} = f_{i-1}$.

case (1)
Assume that $g_i < f_i$; then adding g_i to $f_1 \cdots f_N$ will create a cycle and we can remove any edge $f_i \cdots f_N$ to obtain a better MST contradicting the assumption that $f_1 \cdots f_N$ are the sorted edges in the MST.

case (2)
Assume that $g_i > f_i$, Then consider the sequence of edges $g_1 \cdots g_{i-1}, f_i$. The reason g_i was picked over f_i is that $g_1 \cdots g_{i-1}, f_i$ would cause a cycle. But as $g_1 = f_1 \cdots g_{i-1} = f_{i-1}$, $f_1 \cdots f_{i-1}, f_i$ should cause a cycle contradicting that $f_1 \cdots f_N$ is a MST.

Thus $g_1 \cdots g_N$ is a MST.

Assumption : all edge weights are distinct.

2.3 Consider the weighted directed graph shown in figure 2. Trace through the steps of Dijkstra's algorithm for computing the shortest path from vertex A to all the other vertices. Indicate the vertices in the cloud, and the values in the priority queue at each step. Ties are broken based on alphabetical order.

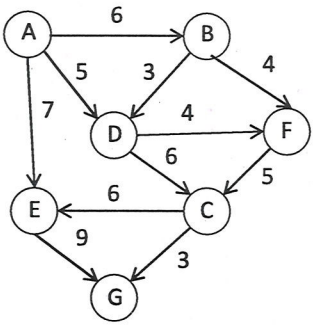


Figure 2: Weighted directed graph for question 2.3

	cloud	outside the cloud (priority queue)
(1)	A 0	D B E 5 6 7
(2)	A D 0 5	B E F C 6 7 9 11
(3)	A D B 0 5 6	E F C 7 9 11
(4)	A D B E 0 5 6 7	F C G 9 11 16
(5)	A D B E F 0 5 6 7 9	C G 11 16
(5)	A D B E F C 0 5 6 7 9 11	G 14.
(6)	A D B E F C G 0 5 6 7 9 11 14.	

2.4 Implement the *Queue operations (enqueue, dequeue, front, and isEmpty)* using *two stacks and the stack operations push, pop, top, and isEmpty.* You cannot use any other Python data structure.

Sin Sout ⟶ dequeue.

↓

inserting (enqueue)

```
Enqueue(o)
Sin.push(o)

Dequeue()
while !sin.isempty():
    s.out.push(sin.pop())
o = sout.pop()
while !sout.isempty():
    sin.push(sout.pop())
return o.

Front()
while !sin.isempty()
    sout.push(sin.pop())
o = sout.top()
while ! sout.isempty()
    sin.push(sout.pop())
return o.

isEmpty()
    return sin.isempty().
```

3 Seven pointers (2 x 7 = 14 points)

3.1 Given an unsorted array A of N distinct integers and an integer k such that $1 \leq k \leq N$ Design an algroithm that returns the k largest elements of A in sorted (Descending) order with the time complexity $O(n + k \log n)$. The algorithm can use only constant memory. You cannot use any Python syntax.

If i is the parent the left child is at $2i+1$ and the right child is at $2i+2$.

Key idea: work backwards from $N/2$ to ensure that array is transformed in to a heap (max heap)

Assuming N is even - bottom up max heap construction.

```
for i = N/2 to 0
    if A[2i+1] > A[2i+2]
        if A[P] < A[2i+1]
            temp = A[i]
            A[i] = A[2i+1]
            A[2i+1] = temp.
        endif.
    Else if A[P] < A[2i+2]
            temp = A[i]
            A[i] = A[2i+2]
            A[2i+2] = temp.
        endif.
    endif.
```

O(n).

```
Downheap(A, L)
    i=0
    while i < L and 2i+1 < L and 2i+2 < L
        if A[2i+1] < A[2i+2]
            if A[P] < A[2i+1]
                temp = A[i]
                A[P] = A[2i+1]
                A[2i+1] = temp.
            endif
        else if A[i] < A[2P+2]
                temp = A[P]
                A[P] = A[2i+2]
                A[2i+2] = temp.
            endif
```

O(klogn).

```
L = N
For i = 0 : K
    A[N-1-i] = A
    Temp = A[N-1-i]
    A[N-1-i] = A[0]
    A[0] = Temp.
    L = L-1
    Downheap(A, L)
```

Total O(n+klogn)

3.2 There is a staircase with n steps. Each step i has a score value $value[i]$. Anand starts at the ground floor 0 and wants to reach the top. On each move, he can climb 1 or 2 steps forward, collecting the score for every step he lands on. He cannot land on two consecutive even-numbered steps. Write a polynomial time algorithm for finding the maximum achievable score.

If i is odd then j can be i+1 or i+2

If i is even then j can be only i+1.

Thus the other way around is

if i is even score(i) = score(i-1) + value(i)

if i is odd score(i) = max(score(i-1), score(i-2)) + value(i)

Using dynamic programming:

score(1) = value[1]

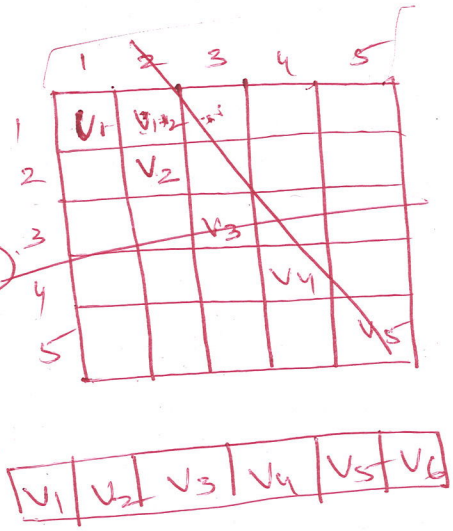
for i = 2 : N

if i is even

score(i) = score(i-1) + value(i)

Else

score(i) = max(score(i-1), score(i-2)) + value[i].



V_1	V_2	V_3	V_4	V_5	V_6

score[0] = 0 score[1] = value[1]

for i = 2 : N

if i is even

score[i] = score[i-1] + value[i]

else

score[i] = max(score[i-1], score[i-2]) + value[i].

max score = score[N-1]

Use this Page for Rough Work

Use this Page for Rough Work