HW4:

(1) (a) ~~We first arrange the elements in ascending order.~~
First, we use half smallest elements for max heap, rest half for min heap.
In this case, the median will be the first element of the max heap.
Hence, this element can be found in constant time O(1).

(b) Extract-Median:

First we extract the biggest element on max-heap.
Now, check ~~If the~~
If the size of maxHeap becomes lesser than the size of min-heap
then extract the smallest element on min-heap and ~~put~~ move it in the max-heap.
EndIf

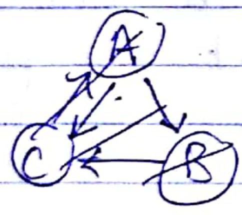
Insert:
Let 'a' be a new element to be inserted.
First check if a is smaller than or greater than or equal to the current median.
If a < current median, then
insert 'a' to max-heap.
else if a $\geq$ current median, then
insert 'a' to min-heap
EndIf
Now, If size of max heap increases to size of min heap by more than ~~0~~ '1',
then insert the max element of max-heap into min-heap
Else If size of min heap increases, then

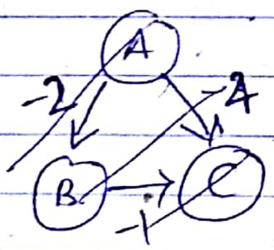
insert the minimum element of min heap
into max heap
End If.

(2)

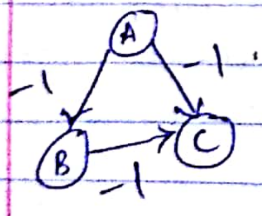
Let ~~an~~ the first integer enter into the server
If
~~Let~~ while (stream of integers != NULL)
Insert the new integer into the server
Apply min-heap to the total elements
inside the server if the new integer
is ~~greater~~ smaller than the last
integer in the array.
Else if the last element in the array
is smaller than the new integer,
keep the new integer into the
new last position.
End If
End while.
Now, extract k largest elements from this
min-heap in constant time, and discard
the rest other elements.

(3)





Here, at first the shortest
path will be ~~between~~
$A \to B \to C \Rightarrow -2$
for A to C.



Now, after making the modification
of adding 1, the shortest path will
~~$A \to B \to C \Rightarrow -2+-2 = -4$~~
be directly from $A \to C$ as well.
So it is $\boxed{\text{false}}$

Scanned by CamScanner

④ $G = (V, E, w)$
Shortest path distances : $\delta(s, u)$
source vertex

We can go in the reverse direction here.
Choose t node first and check all the nodes whose outgoing edges go towards 't' or check all the incoming edges in t and the connected nodes.
~~Now check the shortest distances amongst all edges~~
~~Let the shortest distant node~~
Now check the shortest distance of all the connected nodes to the source node.
So any one or more node will satisfy the condition:

Shortest path from source to t	= Shortest path from source to the connected node of t	+	Shortest path from connected node of t to t.

So the runtime of the algorithm is $O(|V| + |E|)$

⑤ Let us consider a source node s.
Now, we check all the outgoing edges from the source and compare its associate value.
We ~~pick the highes~~ also check the path that is / are possible from the source to the target vertex.
After find those paths, we choose the path with the maximum associated value from one node to the other, till the targeted vertex.
(greedy strategy).

This way we can find the most efficient (reliable) path between given to vertices.
(Here we consider one node as source and the other as

6) **Proof by contradiction:**

Let us consider two distinct minimum spanning trees A and A' in graph G.
Both the trees have same number of edges but all the edges of A need not be same as that of A'.
Let there be an edge e' in tree A' which is not present in A.
Now if we add this edge to tree A, it will form a cycle and hence disrupts the structure of minimum spanning tree.
This proves that the graph G has a unique minimum spanning tree.

7) → We first apply the cycle property 9 time [∵ n+8 edges at most]
→ Apply BFS till a cycle is found.
→ Delete the heaviest edge on this cycle.
→ We repeat this 9 times.

8) Let us consider four vertices with costs 2, 2, 2, 1
In this case, each minimum spanning tree will have 3 edges with an combination other than 2, 2, 2.
The spanning tree with the combination 2, 2, 2 will not be minimum.
So we cannot conclude that T (spanning tree) itself must be a minimum-cost spanning tree in G.