COMP 3804 Assignment 2:

①

— We can get the sorted list in $O(n\log K)$ time by organizing the lists into a minimum heap.

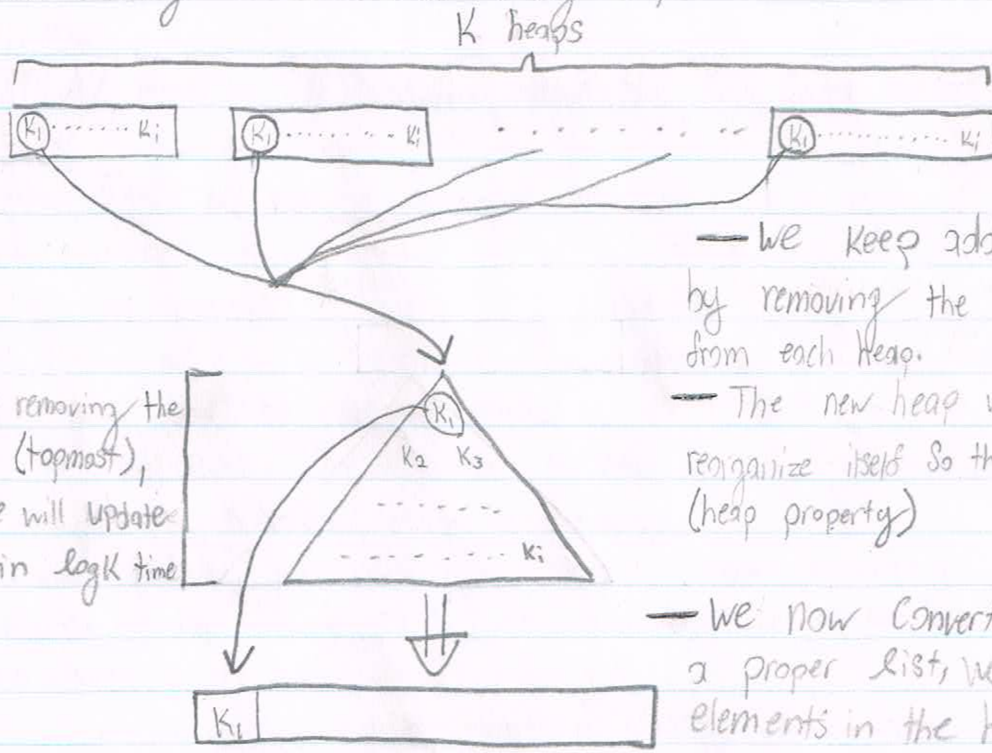
— The first element of each K heap is the smallest element in that heap.

Ⅰ — We must search through all the elements in the heap in order to make the min. heap (n time)

Ⅱ — In order to remove an element from each heap, we must execute a runtime of $\log K$ to reorganize the array

— The product of steps Ⅰ & Ⅱ is $(n)(\log K) = n\log K$ time

Diagram: Getting a sorted list using steps Ⅰ & Ⅱ



— We keep adding elements by removing the first element from each heap.

— The new heap w/ automatically reorganize itself so that its balanced (heap property)

— After removing the root (topmost), the tree will update itself in $\log K$ time

— We now convert the heap into a proper list, we put the topmost elements in the heap to the leftmost position in the list

· The heap will reorganize itself in $\log K$ time, & will always have the smallest element at the root.

— We will repeat the process of removing the root & reorganizing the heap for a total of n times (this is the total # of elements).

— We take n elements, & run a resort on the heap ($\log K$) for each element:

∴ the runtimes the product of the # of elements & the time to rebalence the heap

$n \cdot \log K \rightarrow O(n\log K)$

Psudocode:

```
let minArray be a minimum Sorted array
let heapArray be the the array of all K heaps
let n be the # of elements
let K be the # of heaps
let minHeap be a blank heap
FOR all Integers in n:
    REMOVE the root from a heapArray heap & add it to minHeap
    UPDATE the heap in heapArray we removed the root from
END FOR
FOR all Integers in n:
    REMOVE the root from minHeap & add it to the leftmost free position in minArray
    UPDATE minHeap
END FOR
DISPLAY minArray //Show the output, the Singly Sorted list of all n numbers
```

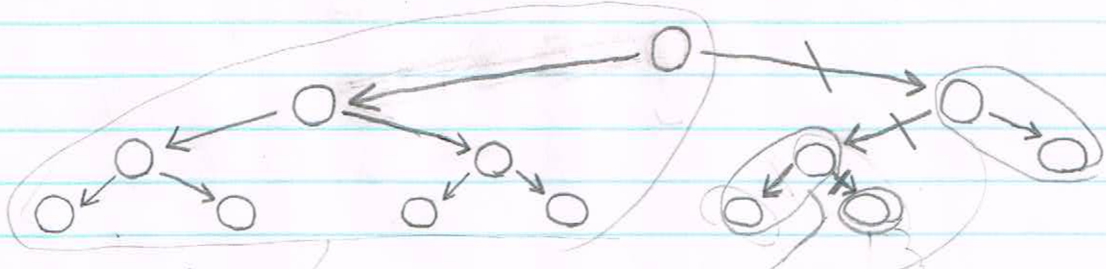
Proof: The Algorithm will Sort the elements in order

- We are always getting the minimum element from each of the K-heaps
  - Each of the these minimal elements will be added to a new heap which is to contain all the n elements
- The heap w/ all n elements will be updated each time a new element is added
  - This ensures that the root w/ always be the minimal element
- We can then remove the root from the heap of all n elements & add the root value to the leftmost position in minArray
  - This means minArray will always have larger & larger elements being added to the end
  - As long as we keep updating the heap every time we remove the root, the root remains the Smallest element in the heap
- To Summerize: We always have the Smallest remaining element added to the array, & ∴ the resulting array will be sorted least to greatest.

(2)

- We are given two input heaps called n-heap & K-heap
  - n-heap: n = 26 elements
  - K-heap: K = 13 elements

K-heap: 13 elements



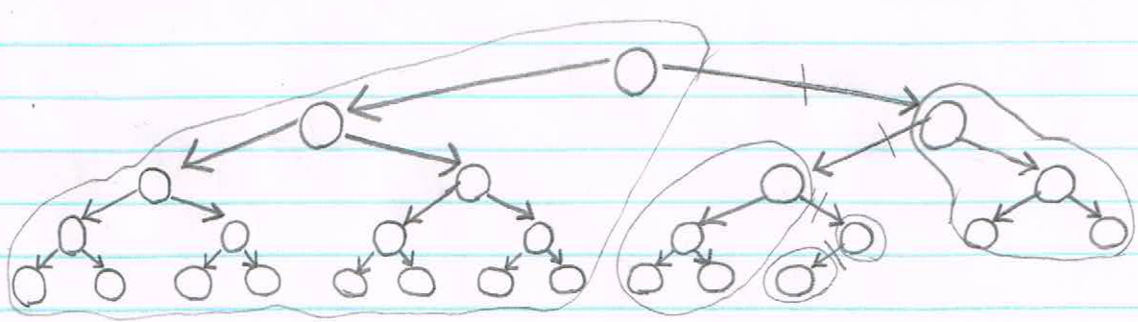
this leaves:

# of remaining Trees:	1	0	2	1
# of nodes:	2^3	2^2	2^1	2^0

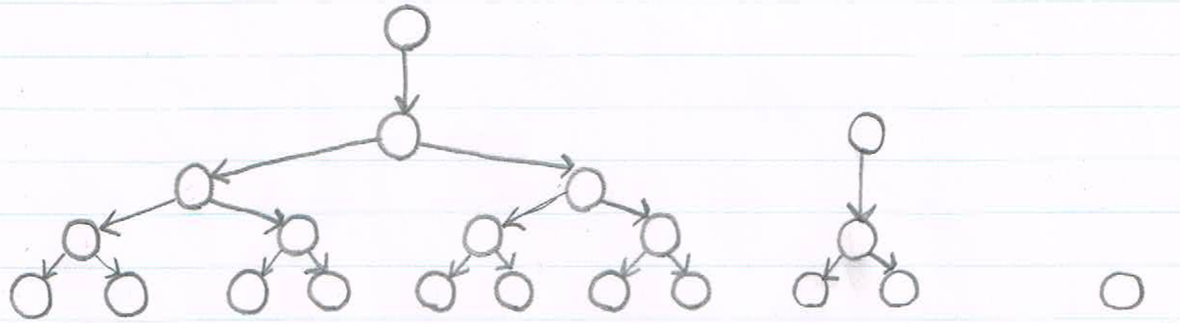
Descriptor: $(1, 0, 2, 1)$

∴ the Descriptor for the K-heap is $(1,0,2,1)$

n-heap: 26 elements



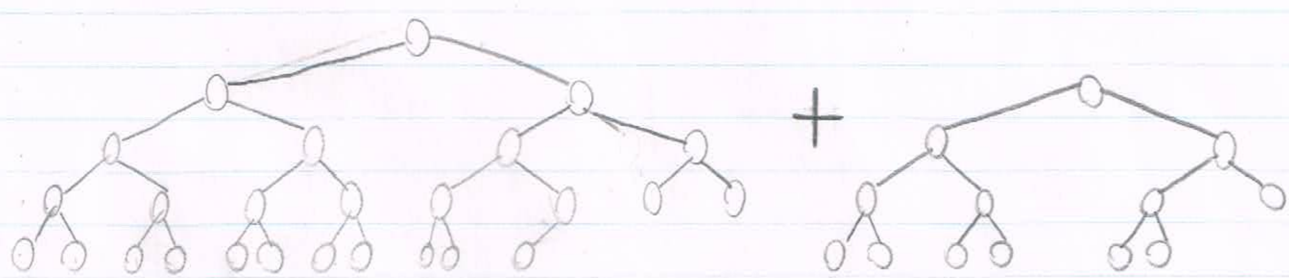
this leaves:



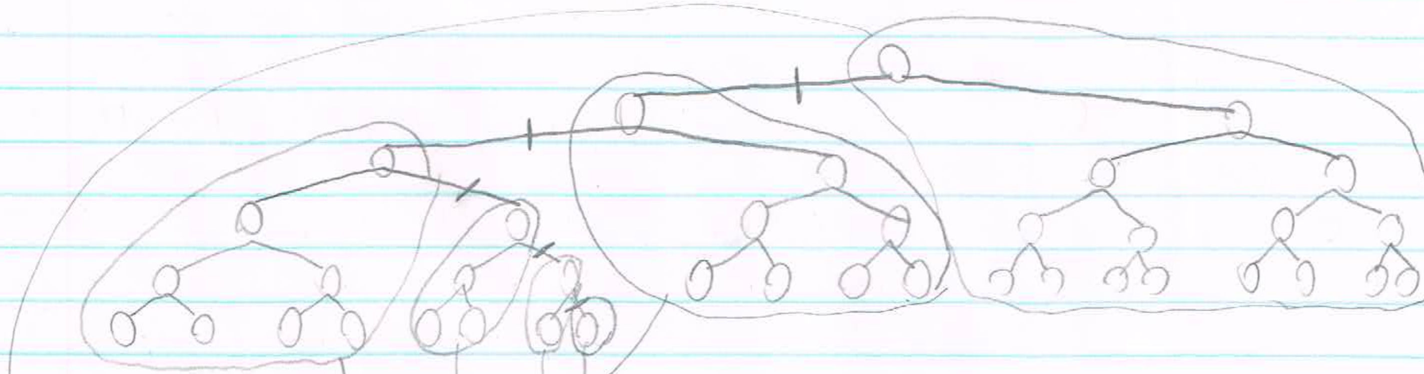
# of remaining Trees:	1	0	2	0	2
# of Nodes:	2^4	2^3	2^2	2^1	2^0
Descriptor:	(1,0,2,0,2)				

$\therefore$ The descriptor for the n-heap is $\boxed{(1,0,2,0,2)}$

Merge the k & n heaps.



this heap is size 39:



$2^4 : 1$

$2^3 : 2$

$2^2 : 1$

$2^1 : 1$

$2^0 : 1$

$\Rightarrow$ Discriptor:

(1,2,1,1,1)

the discriptors are:

the 13 heap is: (1,0,2,1) → (1,1,0,1)
the 26 heap is: (1,0,2,0,2) → (1,1,0,1,0)
the 37 heap is: (1,2,1,1,1)

] merge the nodes to form larger trees

— the 26 heap has 2 trees of 2^0 which sums to 1 node of 2^1.

— The 13 heap has 2 trees of size 2^1 which merge to 2^2.

— The 26 heap has 2 trees of size 2^2 which merge to 2^3.

the sum of the merged discriptors is:

(1,1,0,1) + (1,1,0,1,0) = (1,2,1,1,1)

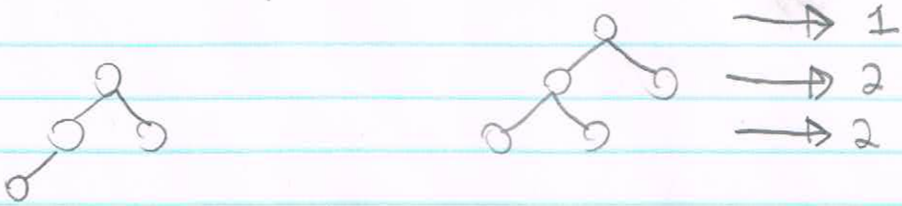
the 37 trees discriptor is: (1,2,1,1,1)

] ∴ the discriptor that arises from merging the two heaps is (1,2,1,1,1)

③ Diagram of heaps: balanced

2^1 2^2 2^3

unbalanced heaps:



- In all cases, the max. or min. element will always be the root.
  - This means for a tree of n elements, 1 element is the root & n-1 elements will have variable placment
  - The root has a fixed placement
- For a heap of size n:
  - No matter what the heap contains, the nodes will always form the same tree connectivity (shape) due to the heap property that smaller elements are children of larger ones (for a max. heap)
- With the root node as a fixed position, it will have two subtrees as children. Let these subtrees be named A (left subtree) & Z (right subtree)
  - the sum of A+Z will equal all the children of the root
  - the root plus its children equal all the elements in the tree
  - ∴ n (all elements) − 1 (the root) = A+Z

$$A+Z=n-1$$

- there are always a fixed # of tree permuta. per any tree size. Let this # of permutations be n, the total # of heaps.
  - Thus; T(n) is the # of unique heaps, to n
- All elements in the leftmost subtree (A) are smaller then the root
  - let the # of nodes in the left subtree be α
  - there are n-1 elements in A+Z, if $A=\alpha$ then $Z=n-1-\alpha$
- to sumarize, we know that:
  - n = # of elements
  - A = left subtree from root
  - Z = right subtree from root
  - $A+Z=n-1$
  - $A=\alpha$
  - $Z=n-1-\alpha$
  - T(n) = # of unique heaps for heap size n elements

- the left subtree, A, is smaller then the root & $\therefore$ we must count the # of permutations it may hold:
  - A has a max of $n-1$ elements (when $Z=0$)
  - thus, if P is the current permutation of the left tree, there are $\binom{n-1}{\alpha}$ different nodes we may pick in the left tree
  - let P = permutation of heap

    let α = elements in A

    then we may pick α # of elements in $\binom{n}{\alpha}$ elements

- lets build a recurence:

  $T(1) = 1$

  $T(2) = 1 \cdot T(1)$ → only 1 choice for the leftmost tree

  $T(3) = 1 \cdot T(2) \cdot T(3)$

  $T(4) = 3P_2 \cdot T(2) \cdot T(1)$ → $\alpha = 2$ & theres 3 children

  ⋮

  $T(n) = ({}^{n-1}P_\alpha)(T(\text{left recursion}))(T(\text{right recursion}))$

  → tree A has α elements & tree Z has $n-1-\alpha$ elements

  $\therefore$ $T(n) = \binom{n-1}{\alpha}(T(\alpha))(T(n-1-\alpha))$

④ a) we Start from edge A. Prims Alg.:

Choose: $\boxed{6/D}$

Set S	A	B	C	D	E	F
Nil	0	∞	∞	∞	∞	∞
A	V	8/A	5/C	∞	1/E	∞
A, E	V	7/B	7/C	$\boxed{6/D}$	V	1/F
A, E, F	V	∞	2/6	8/D	V	V
A, E, F, C	V	∞	V	∞	V	V
A, E, F, C, D	V	3/B	V	V	V	V
A, E, F, C, D, B	V	V	V	V	V	V

no Path, → move back to next Shortest Path.

— Circles represent the Shortest Path from the Current tree,
· Example: A, E has 1/F, this means the Shortest Path from the A, E tree is to move to F.
— Nodes that are marked V in the right table have already been visited.
— ∞ means there is no available Path

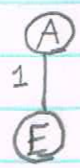
total weight of the minimum Search tree is: 1+1+6+2+3=13

Spanning-Tree Step-by-Step:

— the Subtree must Contain the Smallest # of edges needed to Connect all verticies. Always Choose the lowest weighted edge that Connects a visited & nonvisited vertices.
— A graph Should Contain n-1 verticies for n edges, as that represents the minimal # of Connections.

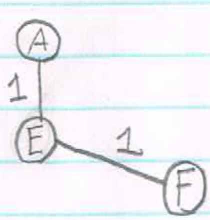
Nil: Move to A (its the Starting Node)
- we must Start w/ node A, per the assignment instructions

A: Move to E (from A)
- we may move to neighbours B, E, or F
- A → is minimum w/ a weight of 1:



A → B → 8
A → C → 5
A → E → 1 minimum

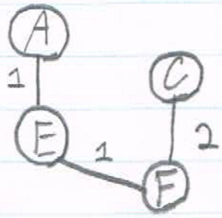
A, E: Move to F (from E)
- Neighbours to E: A, B, C, D, F
· A is alredy visited
· F has minimal weight of 1, move to F



A → B → 8
A → C → 5

E → C → 7
E → D → 6
E → F → 1 minimum
E → B → 7

A, E, F: Move to C (from F)
- Neighbours include C, E, & D. F is alredy visited & is excluded
· C is lowest weight, include it:



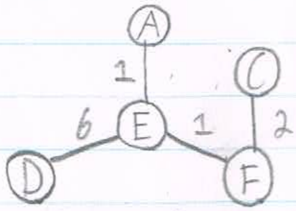
A → B → 8
A → C → 5

F → D → 8
F → C → 2 (minimum)

E → C → 7
E → B → 7
E → D → 6

A, B, C, F: Move to D (from E)

- C has neighbours of A, E, & F. All neighbours have been visited, thus we must move back to the next lowest weight node.
- move to node E:
  - E may connect to A, B, C, D, & F. A, B, C, & F are already visited.
  - D is the neighbour w/ the lowest weight, include it first:

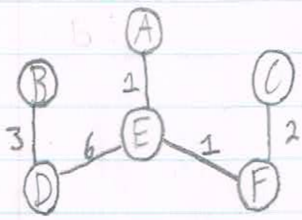


A → B → 8
F → D → 8

E → D → 6 minimum
E → B → 7

A, B, C, F, D: Move to B (from D)

- D has neighbours of F, E, & B: F & E have already been visited
- include B to the graph:



D → B → 3 minimum
A → B → 8
E → B → 7

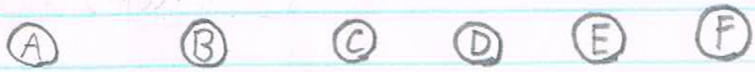
- The total weight of the tree is:
  1 + 1 + 2 + 6 + 3 = 13

Kruskal's Algorithm:

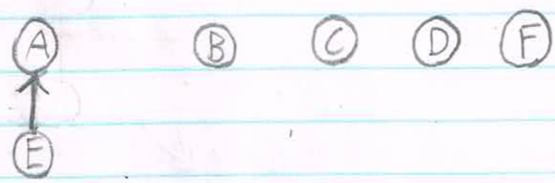
- We sort edges by weight (lowest first)
  - We select the smallest weight edge to add to the min. spanning tree
  - We must then choose the next lowest weighted edge, such that the tree doesn't form a cycle. Add this to the tree.
- We continue adding the lowest weight edges until n-1 edges have been added for the n nodes in the graph.
- We must sort the lines from the highest priority (low weight, closeness to A) to the lowest priority (highest weight, closeness to Z)

line	Edge	Weight	line	Edge	Weight
1	AE	1	6	DE	6
2	EF	1	7	BE	7
3	CF	2	8	CE	7
4	BD	3	9	AB	8
5	AC	5	10	DF	8

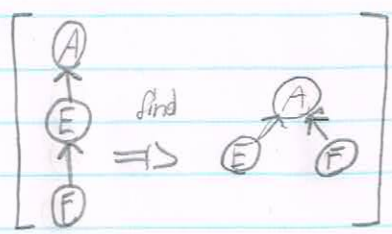
Path Compression Steps: We Start at A



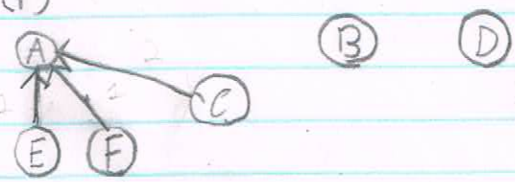
Ⅰ Union (E, A)
- Find (E)
- Find (A)

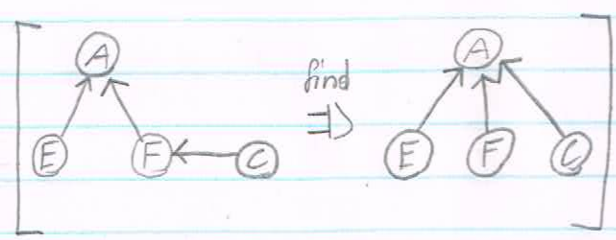


Ⅱ Union (F, E)
- Find (F)
- Find (E)

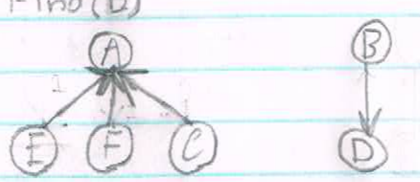


Ⅲ Union (C, F)
- Find (C)
- Find (F)

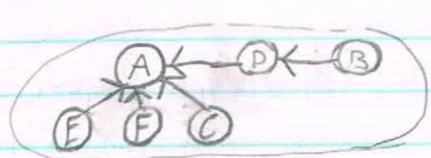


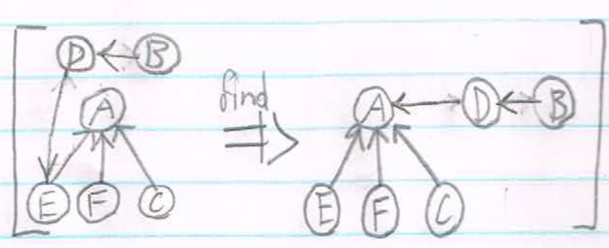


Ⅳ Union (B, D):
- Find (B)
- Find (D)



Ⅴ Union (D, E)
- Find (D)
- Find (E)

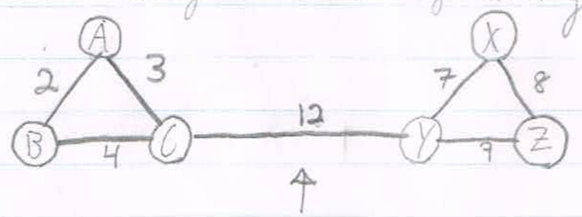




(5) This can generalize to all other graphs!

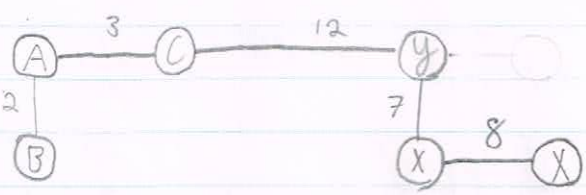
a) $G_1 = (V_1, E_1)$] both graphs are disconected
$G_2 = (V_2, E_2)$] G is a graph for $V = V_1 \cup V_2$ & $E = E_1 \cup E_2$

i) If we modify G_1 & G_2 by adding an edge, we get $G = (V, E)$ as follows:



G_1 & G_2 are joined w/ an edge of height 12

the MST is as follows: (Start from A)



the minimum serch is:
$2+3+7+8-12 = 32$

Proof: using Kruskal's Algorithm:

Pair	Edge	Weight
1	AB	2
2	AC	3
3	BC	4
4	xy	7
5	XZ	8
6	yz	9
7	cy	12

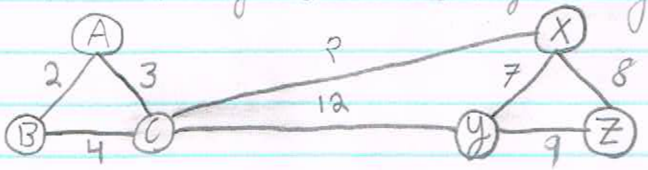
Start at A
Add AB → 2
Add AC → 3
Ignore BC → cycle
Add XY → 7
Add XZ → 8
Ignore YZ → cycle
Add CY → 12
Stop → we have n-1 edges, the MST's complete

total MST:
$2+3+7+8+12 = 32$

∴ The MST above is correct

(*) — as can be seen, the MST of G must always contain the edge connecting G_1 & G_2, otherwise the MST won't be fully connected.

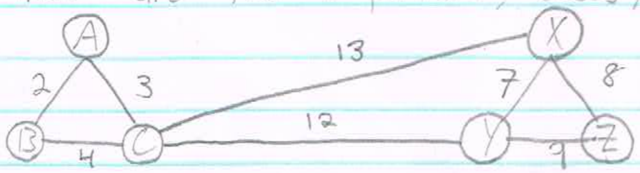
ii) If we modify G_1 & G_2 by adding two edges, we get $G \doteq (V, E)$ as follows



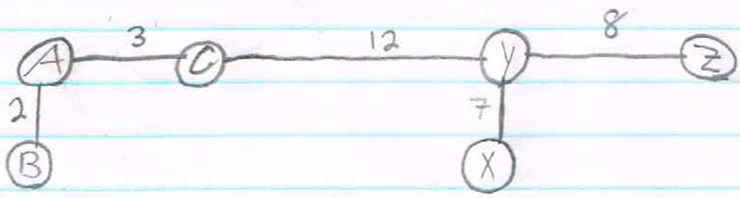
P = any Natural #

Case 1:

there are two important cases, if $P \geq 8$ (lets say $P = 13$):



the MST is as follows:



The Min. Search is: $2+3+7+8+12 = 32$

Proof: use Kruskal's Algorithm

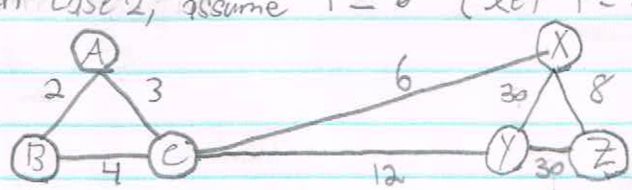
Pair	Edge	Weight	
			START at A
1	AB	2	ADD AB → 2
2	AC	3	ADD AC → 3
3	BC	4	Ignore BC → cycle
4	XY	7	ADD XY → 7
5	XZ	8	ADD XZ → 8
6	YZ	9	IGNORE YZ → cycle
7	CY	12	ADD CY → 12
8	CX	13	STOP → n-1 edges, MST is Complete

total MST:
$2+3+7+8+12 = 32$
∴ the MST proves correct

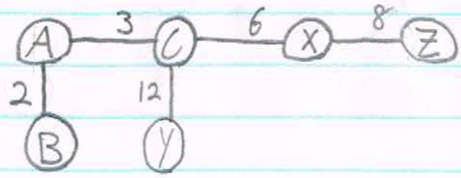
In this case, the newly added edge of ($P=13$) weight 13 is not in the MST tree, as its value is greater then 7, meaning its skipped.

Case 2:

In case 2, assume $P \leq 6$ (let $P = 5$), & set XY & ZY to 30



The MST is as follows:



the Min. Serch tree is:
$2+3+6+8+12 = 31$

Proof via Kruskals Alg.:

Pair	Edge	Weight
1	AB	2
2	AC	3
3	BC	4
4	CX	6
5	XZ	8
6	CY	12
7	XY	30
8	YZ	30

START at A
ADD AB → 2
ADD AC → 3
IGNORE BC → Cycle
ADD CX → 6
ADD XZ → 8
ADD CY → 12
STOP → n-1 edges, MST is Complete

total MST:
$2+3+6+8+12 = 31$

∴ The MST above is Correct

In this Case, the newly added edge of (P=6) weight 6 is in the MST tree because its part of the Min. Search.

∴ Both edges may be part of the MST for G.

Results were Proven via. Kruskal's Alg.

(*) • As we can seen in Case 2, if the two edges can be in the MST if they are both the lowest weight edges to a Specific node.
• As Seen in Case 1, its possible that only 1 edge is in the MST, because although we need atleast 1 edge to Connect the Subgraphs, the Second edge is only added When its the minimum weight edge to a Specific node.

b) A G has a MST if:

If the Alg. Simply treats all edges as having weight 1, we can ignore this property

Ⓘ • If it is Connected (theres atleast 1 MST) & G is Weighted

- If its disconnected or its unweighted, its not Possible to determine a Path to connect all the nodes

Ⓘⓘ • If the G is undirected & Ⓘ is met, it must have a MST

- There is Certainly a way to Connect all nodes, as theres no restrictions on how the MST may form.

Ⓘⓘⓘ • If it directed & Ⓘ is met, then the directedness Cannot force cycles in the MST & Cannot make any nodes unreachable

- by definition, A MST Cannot have a cycle & must contain all n nodes
- undirected graphs have a maximum of n^{n-2} Spanning trees w/ a minimum of 1 MST

Ⓘⓥ • If all the edges of G have unique weights, there is 1 unique MST, otherwise there may be more then one MST

Ⓥ • The G must have more then one node

- Otherwise, the "MST" Cannot be a tree, as theres only one node c)

Ⓐ — let T be the MST for G

Ⓑ — Deleting the heaviest edge dosn't alter T

— Given conditions Ⓐ & Ⓑ, what can we say about G?

- We can say three things:
  1. G contains more then the minimum # of edges for connectedness (n-1)
  2. G is a connected Graph
  3. We know that the heaviest edge dosn't connect to a vertex of degree 1
  4. The heaviest edge is part of a cycle

— Prove ①:

- Case 1: G has less then n-2 edges
  - the Graph is disconnected, & ∴ there will be no way to make a MST containing all n nodes
- Case 2: G has n-1 edges
  - If the Graph has the minimal # of edges to be connected, then removing any edge w/ disconnect the graph. A MST can only be formed from a connected graph (see 5.b for proof)

— Prove ②:

- G is connected, by definition a MST must contain all n nodes. If G is disconnected, a MST cannot span all n elements. Thus, in general, G cannot be disconnected in any case.

— Prove ③:

- If the heaviest edge connects to a vertex of degree 1, then removing the edge will disconnect the graph. This makes finding the MST impossible (See 5.b for more proof)

— Prove ④:

- According to the cut property, removing the largest edge from a cycle wont change a MST.
- We remove the heaviest edge from the G, & the MST dosn't change. This may only occur by way of the cut property.

∴ The heaviest edge is part of a cycle

d)

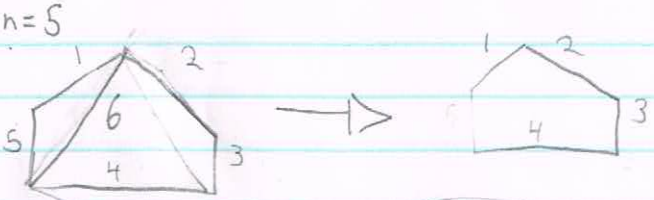
$G = (V, E)$

Edges weights increment by 1 untill $|E|$

- The edges weights are unique $\therefore$ the MST is also unique
- The MST will have a min weight of ~~of~~ all the edge weights up to $n-1$. That is, the Sum of all weights minus the largest weight, n.
- The MST must have $n-1$ edges
- Min weight formula:

$$1+2+3+\dots+n-2+n-1 = \frac{(n-1)(n)}{2}$$

Ex.: let $n=5$



$1+2+3+4+5 = 6+4 = 10$

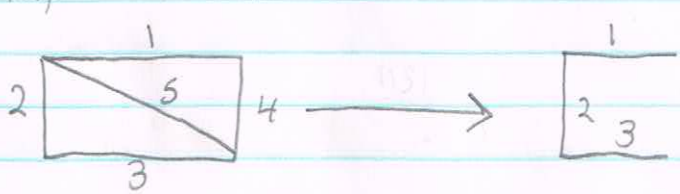
$\frac{(n-1)(n)}{2} = \frac{(4)(5)}{2} = 10$

$\therefore$ This is the lowest MST Possible for this type.

$\therefore$ The lowest min. weight is: $\frac{(n-1)(n)}{2}$

- Does all G have this min. weight: NO

Example: let $n=4$



MST: $1+2+3 = 6$

lowest Possible MST: $\frac{(n-1)(n)}{2}$

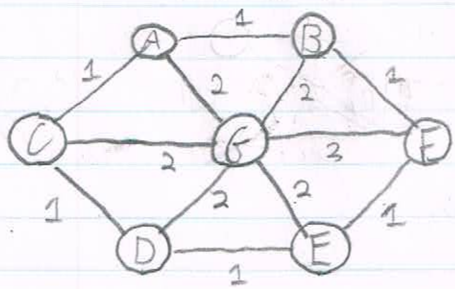
$= \frac{2 \cdot 3}{2} = 3$

$\therefore$ No, its Possible that the MST is larger then the Min. MST Possible for edges $1, 2, 3, \dots, |E|$ in $G = (V, E)$

e) DT is incorrect

- Dijkstra's Algorithm Produces a G of edges representing the Shortest Path from the Source to all nodes in the G.
  - The outputs ∴ a Spanning tree of G
- Assuming the Spanning tree produced by Dijkstra's Alg. is the MST:

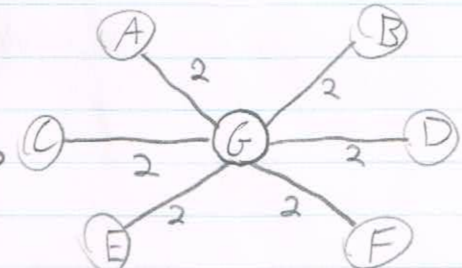
Let G be:



- Set G to be the Start
- find the Shortest Path to all nodes w/ dijkstras Alg.

Dijkstra's Alg.:

Round	A	B	C	D	E	F	G
1	∞,-	∞,-	∞,-	∞,-	∞,-	∞,-	0,-
2	2,G	2,G	2,G	2,G	2,G	2,G	
3		2,G	2,G	2,G	2,G	2,G	
4			2,G	2,G	2,G	2,G	
5				2,G	2,G	2,G	
6					2,G	2,G	
7						2,G	

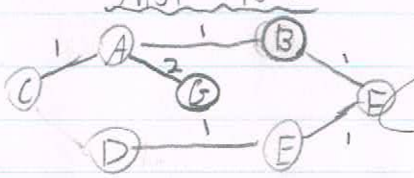
⟹ 

total weight = 12

the real MST is: use Kruskals Alg.

Edge	Weight
AB	1
AC	1
BF	1
FE	1
ED	1
DC	1
AG	2
BG	2
FG	2
EG	2
DG	2
CG	2

MST is:

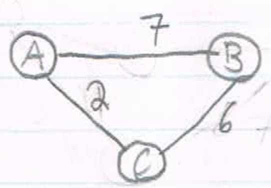
⟹ 

total weight = 7

∴ Dijkstra's Alg. generally won't find the MST

8) DT is incorrect

- The Shortest Path tree is not necisarily the MST when using this Algorithm
- take the following graph:



- Assume A is the Start
  - Goto B

Kruskruls Spanning tree to C is:

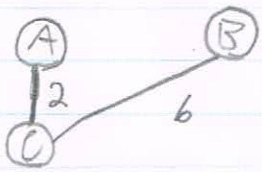
Iteration	A	B	C
1	0,-	∞,-	∞,-
2		7, A	2, A
3		7, A	

⇒

the MST (via. Kruskuls Alg.):

Edge	weight
AB	7
AC	2
BC	6

⇒



MST ⇒ weight of 8

as Seen, the MST does not Contain AB ∴ a Spanning tree from Kruskruls Alg. is NOT necisarily Part of the MST.

(6) Dijkstra's Algorithm: Start at S

iteration	S	a	C	u	V	W	Z
0	0,–	∞	∞	∞	∞	∞	∞
1	0,–	13,S	40,C	∞	∞	∞	∞
2			16,a	113,a	19,a	∞	17,a
3				13,a	18,C	24,C	17,a
4				37,Z	18,C	18,Z	
5				28,W		18,Z	
6				28,W			

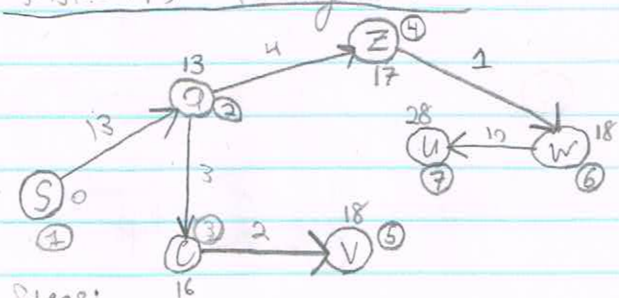
order of build:

S → a → C → Z → V → W → U

Note:

⇒ V & W are both 18 in iterations 4 & 5, either can be picked. I chose V, its the lowest letter.

Dijkstra's Spanning tree:



Values from S:

a ⇒ 13

C ⇒ 16

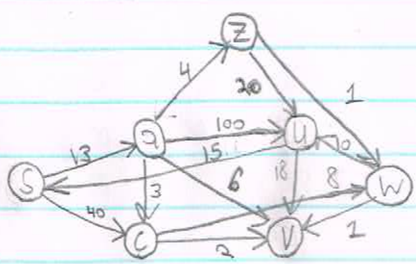
U ⇒ 28

V ⇒ 18

W ⇒ 18

Z ⇒ 17

Steps:



Start at S

iteration 1: S

Add a-13 & C-40 to the Priority queue

ADD edge a-13 & remove it from the Priority queue

• Queue: C-40

iteration 2: S, a

Add edges Z-17, V-19, U-113, C-16 to the Priority queue

ADD edge C-16 from the queue

• queue: ~~C-40~~, Z-17, V-19, U-113

iteration 3: S, a, C

Add edges 13-U, 18-V, 24-W, & 17-Z to the queue

Add edge 17-Z from queue

• queue: 13-U, 18-V, 24-W

iteration 4: S, a, C, Z

Add edges 37-U, 18-V, 18-W to queue

Add edge 18-V from queue

queue: 37-U, 18-W

iteration 5: S, a, C, Z, V

Add edges 28-U to queue

Add edge 18-W from queue

queue: 28-U, 37-U

iteration 6: S, a, C, Z, V, W

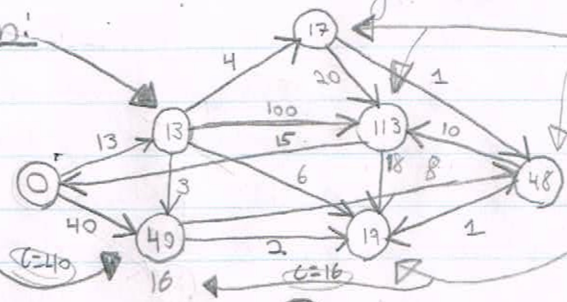
Add edge 28-U from queue → all nodes in Spanning tree: S, a, C, Z, V, W, U

Bellman—Ford:

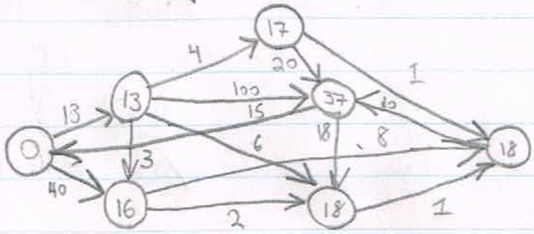
- We start from S in the following graph
- We relex each edge (|E|=n-1) a total of (n-1 = 6) 6 times

first relaxation:

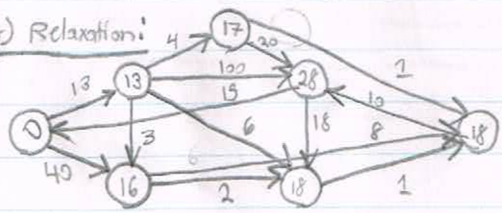
Second relaxation:



third relaxation:



fourth (& Subsequent) Relaxation:



- this is the final relax that results in any change. After relaxation # 4, we get the same Graph for subsequent relaxations.

	0	1	2	3	4	5+
S	0	0	0	0	0	0
a	∞	13	13	13	13	13
C	∞	40	16	16	16	16
u	∞	∞	113	37	28	28
V	∞	∞	19	18	18	18
W	∞	∞	48	18	18	18
Z	∞	∞	17	17	17	17

final answers ⇒

final values

0 ⇒ S
13 ⇒ a
16 ⇒ C
28 ⇒ u
18 ⇒ V
18 ⇒ W
17 ⇒ Z

Spanning tree: