Amortized Cost Analysis

Ex. 1

```
for i = 1 to n
    push or Pop
endfor
```

push takes O(1)
Pop takes O(1)

Our worst case run time complexity = O(n)

Ex. 2

```
for i = 1 to n
    push or pop or multipop
endfor
```

O(n)

push takes O(1) worst-case
pop takes O(1) worst-case
multipop takes O(n) worst-case

Our worst case run time complexity = $O(n^2)$

Aggregate Analysis

- We show that a sequence of n operations (for all n) takes worst-case time $T(n)$ total.

- So, in the worst case, the amortized cost (average cost) per operation will be $T(n)/n$

observation: Multipop takes $O(n)$ time if there are n elements pushed on the stack

A sequence of n pushes takes $O(n)$
multipop takes $O(n)$ worst-case

$T(n) = O(n)$
when amortized over n operations,
average cost of an operation $= O(n)/n = O(1)$

Ex. 2

```
for i = 1 to n                    O(1)
    push or pop or multipop       O(1)   O(1)
end for
```

push takes $O(1)$ amortized

pop " $O(1)$ "

multipop " $O(1)$ "

our worst case runtime complexity = $\Theta(n)$

Accounting Method

- We assign different charges (amortized costs) to different operations

- If the charge for an operation exceeds its actual cost, the excess is stored as credit.

- The credit can later helps pay for operations whose actual cost is higher than their amortized cost.

- Total credit at any time = total amortized cost − total actual cost

Total
- ^Credit can never be negative.

Ex. 2 for $i = 1$ to n

push or pop or multipop

endfor

try # 1

assign a ~~cost~~ charge of (1) to each operation

op.	charge	Actual cost	total credit
→ push	1	1	0
push	1	1	0
multipop	1	2	-1

try # 2

assign charges as follows:

push (2) → $O(1)$

pop 0 → $O(1)$

multipop 0 → $O(1)$

op.	charge	Actual cost	total credit
push	2	1	1
push	2	1	2
multipop	0	2	0

Ex. 2

```
for i = 1 to n                    O(1)
    push or pop or multipop       O(1)  O(1)
end for
```

	Amortized cost
push	$O(1)$
pop	$O(1)$
multipop	$O(1)$

worst-case runtime complexity = $O(n)$

- Fibonacci heaps are loosely based on binomial heaps.

- A Fibonacci heap is a collection of min-heap trees similar to Binomial heaps, however, trees in a Fibonacci heap are not constrained to be binomial trees. Also, unlike binomial heaps, trees in Fibonacci heaps are not ordered.

- Link to Fibonacci heap animations:

www.cs.usfca.edu/ngalles/JavascriptVisual/FibonacciHeap.html

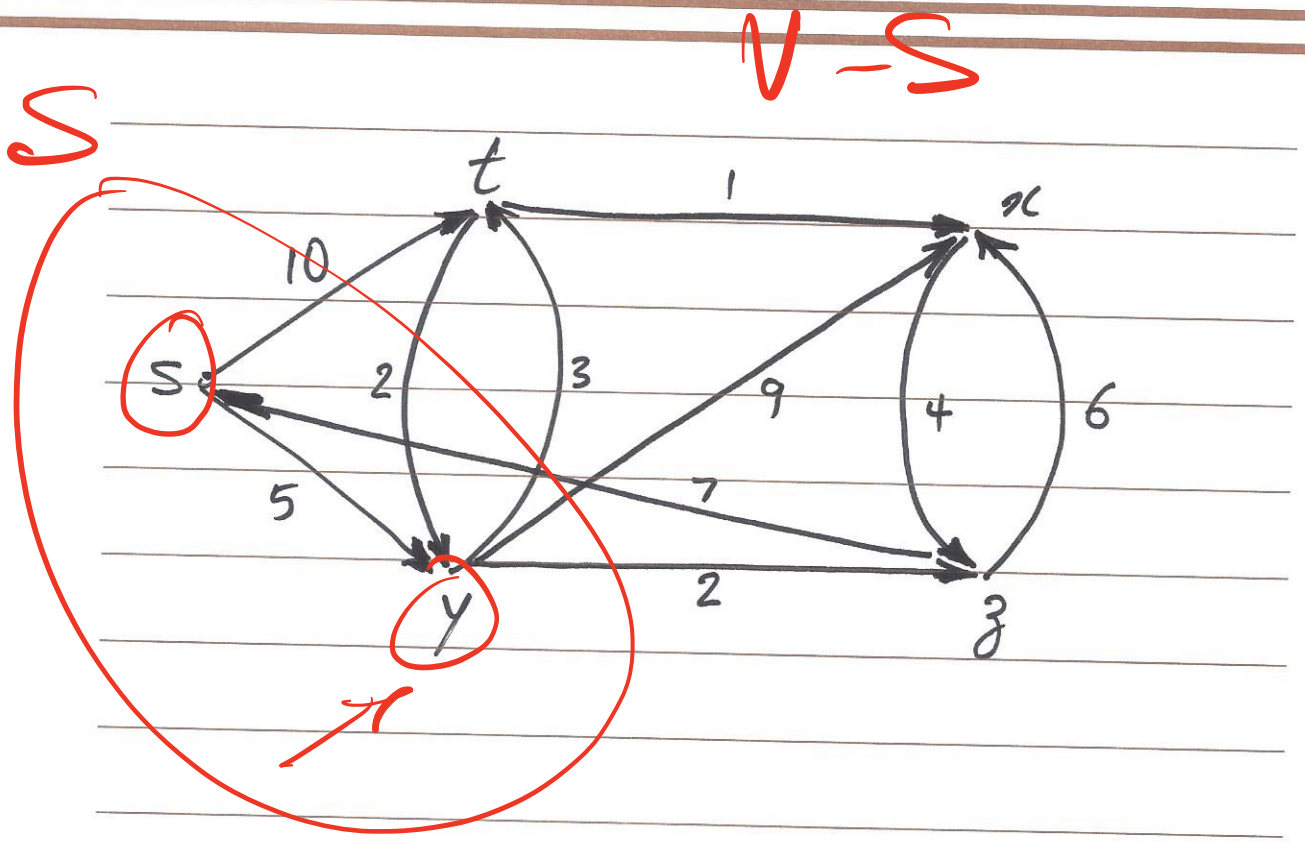
Amortized costs

	Binary Heap	Binomial Heap	Fibonacci Heap
Find-Min	$O(1)$	$O(\lg n)$	$O(1)$
Insert	$O(\lg n)$	"	$O(1)$
Extract-Min	"	"	$O(\lg n)$
Delete	"	"	$O(\lg n)$
Decrease-key	"	"	$O(1)$
Merge	$O(n)$	"	$O(1)$
Construct	$O(n)$	$O(n)$	$O(n)$

Shortest Path

Problem Statement:

Given $G = (V, E)$ with $w(u,v) \geq 0$ for each edge $(u,v) \in E$, find the shortest path from $s \in V$ to $V - s$.

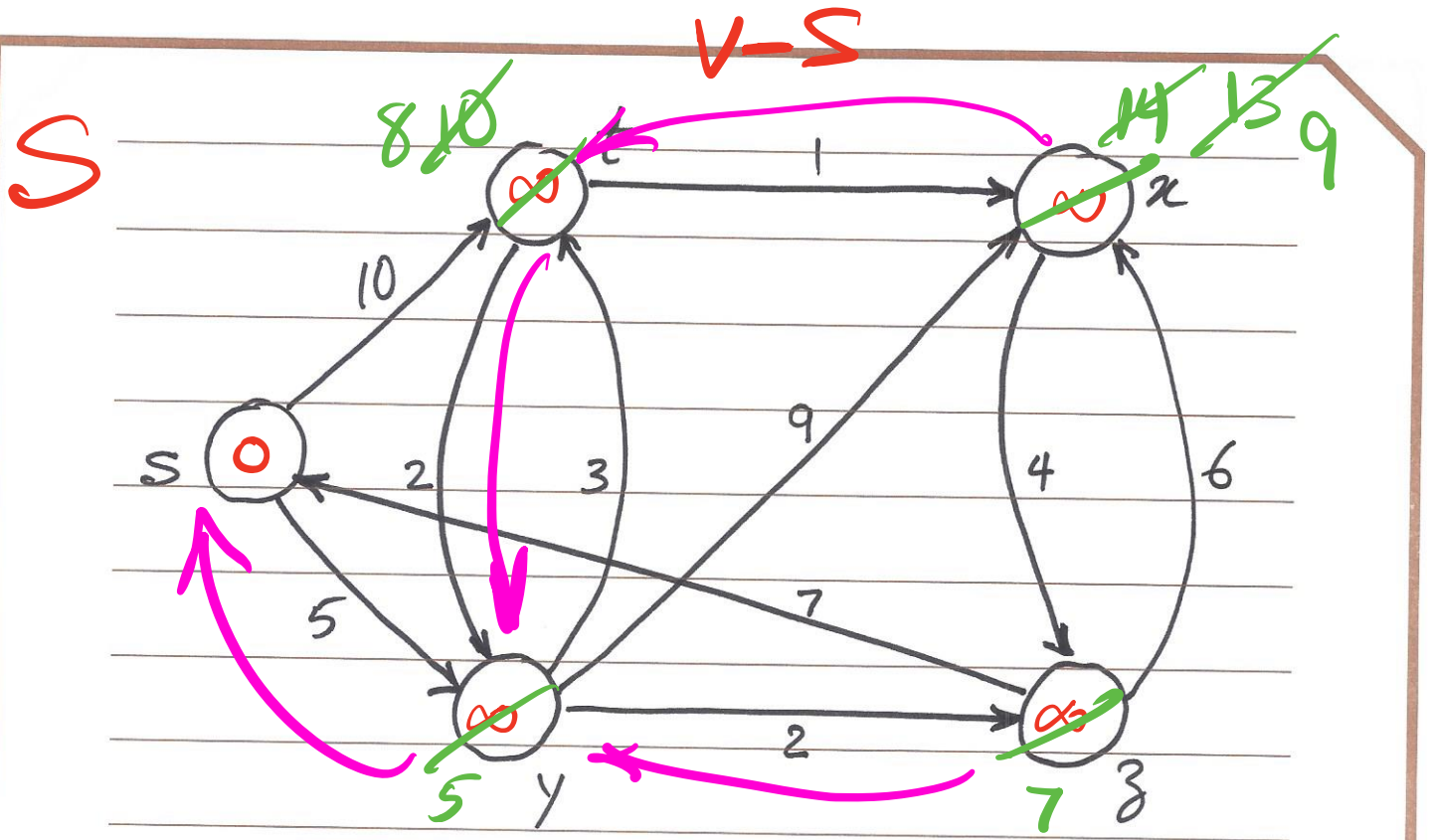


Solution

1. Start with a set S of vertices whose final shortest path we already know.

2. At each step, find a vertex $v \in V - S$ with shortest distance from s.

3. Add v to S, and repeat.



$S_1 = \{s\}$, $S_2 = \{s, y\}$, $S_3 = \{s, y, z\}$

$S_4 = \{s, y, z, t\}$, $S_5 = \{s, y, z, t, x\}$

Dijkstra's Alg.

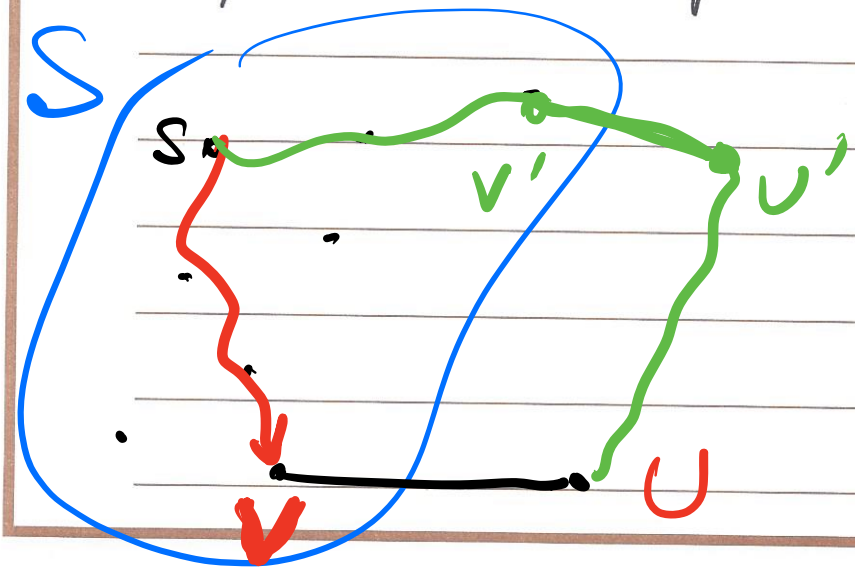
Proof of Correctness

We will prove that at each step, Dijkstra's algorithm finds the shortest path to a new node in the graph.

Proof by mathematical induction:

Base Case: $|S| = 1$, $S = \{s\}$ and $d(s) = 0$

Inductive Step: Suppose the claim holds when $|S| = k$ for some $k \geq 1$. We now grow S to size $k+1$ and prove that we have found the shortest path to the new node.



Implementation of Dijkstra's

Initially $S = \{s\}$ and $d(s) = 0$

for all other nodes $d(u) = \infty$

While $S \neq V$

Select a node $v \notin S$ with at least one edge from S for which

$$d(v) = \min_{e(u,v) : u \in S} (d(u) + l_e)$$

Add v to S

endwhile

More Detailed Implementation of Dijkstra's

S = Null

Initialize priority Queue Q with all nodes V where d(v) is the key value.
(All d(v)'s are $=\infty$, except for s where d(s)=0)

While $S \neq V$
  v = Extract_Min(Q)
  S = S ∪ {v}
  for each vertex u ∈ Adj(v)
    if d(u) > d(v) + le;
      Decrease_key(Q, u, d(v)+le)
  end for
end while

O(n)

O(n)

relaxation step

Decrease key "p"

change u's parent

no of Decrease key "p"

$O(n^2)$ → $O(m)$

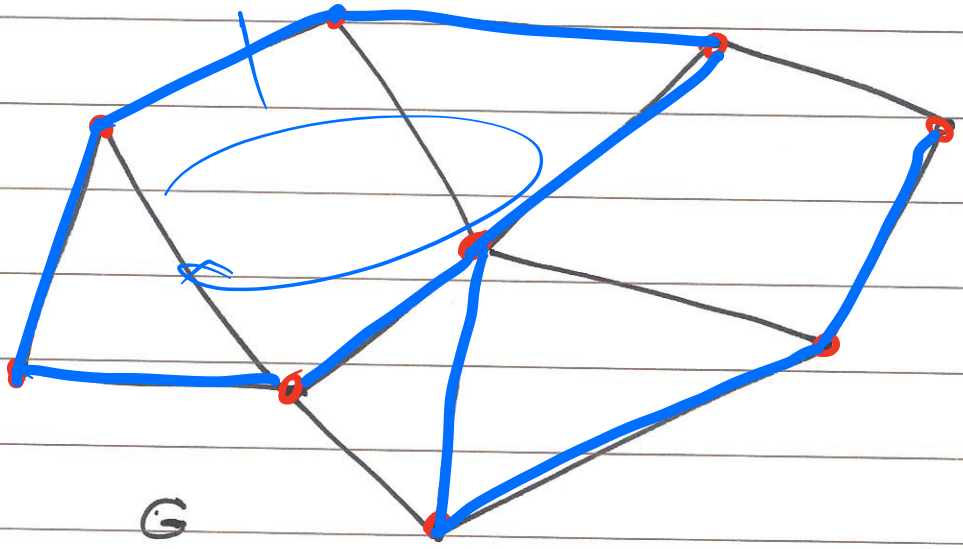
Complexity Analysis

- Initialize Priority Queue : $O(n)$
- Max. no. of Extract-Min op's : n
- Max. no. of Decrease-key op's : m

	Binary Heap	Binomial Heap	Fibonacci Heap
n. Extract_Min's	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$
m. Decrease-key's	$O(m \lg n)$	$O(m \lg n)$	$O(m)$
Total	$O(m \lg n)$	$O(m \lg n)$	$O(m + n \lg n)$
Sparse graph	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$
Dense graph	$O(n^2 \lg n)$	$O(n^2 \lg n)$	$O(n^2)$

Problem Statement

Find minimum cost network that connects all nodes in G.



G

Def. Any tree that covers all nodes of a graph is called a spanning tree.

Def. A spanning tree with minimum total edge cost is a minimum spanning tree. (MST)

Problem Statement

Find a MST in an undirected graph

Sol. 1: Sort all edges in increasing order of cost. Add edges to T in this order as long as it does not create a cycle. If it does, discard the edge.

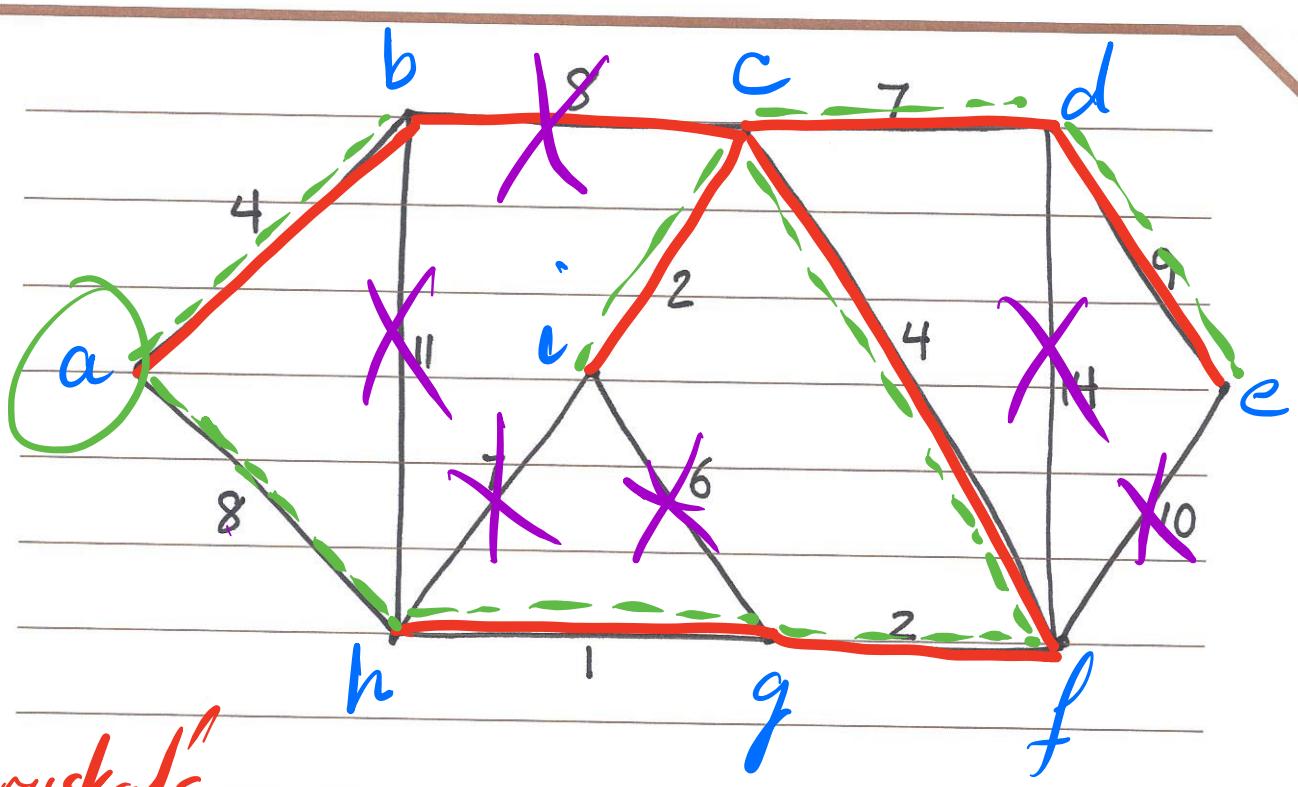
Kruskal's Alg.

Sol. 2: Similar to Dijkstra's algorithm, start with a node set S (initially the root node) on which a minimum spanning tree has been constructed so far. At each step, grow S by one node, adding the node v that minimizes the attachment cost.

Prim's Alg.

Sol. 3: Backward version of Kruskal's.
Start with a full graph (V, E).
Begin deleting edges in order of decreasing cost as long as it does not disconnect the graph

Reverse Delete Alg.

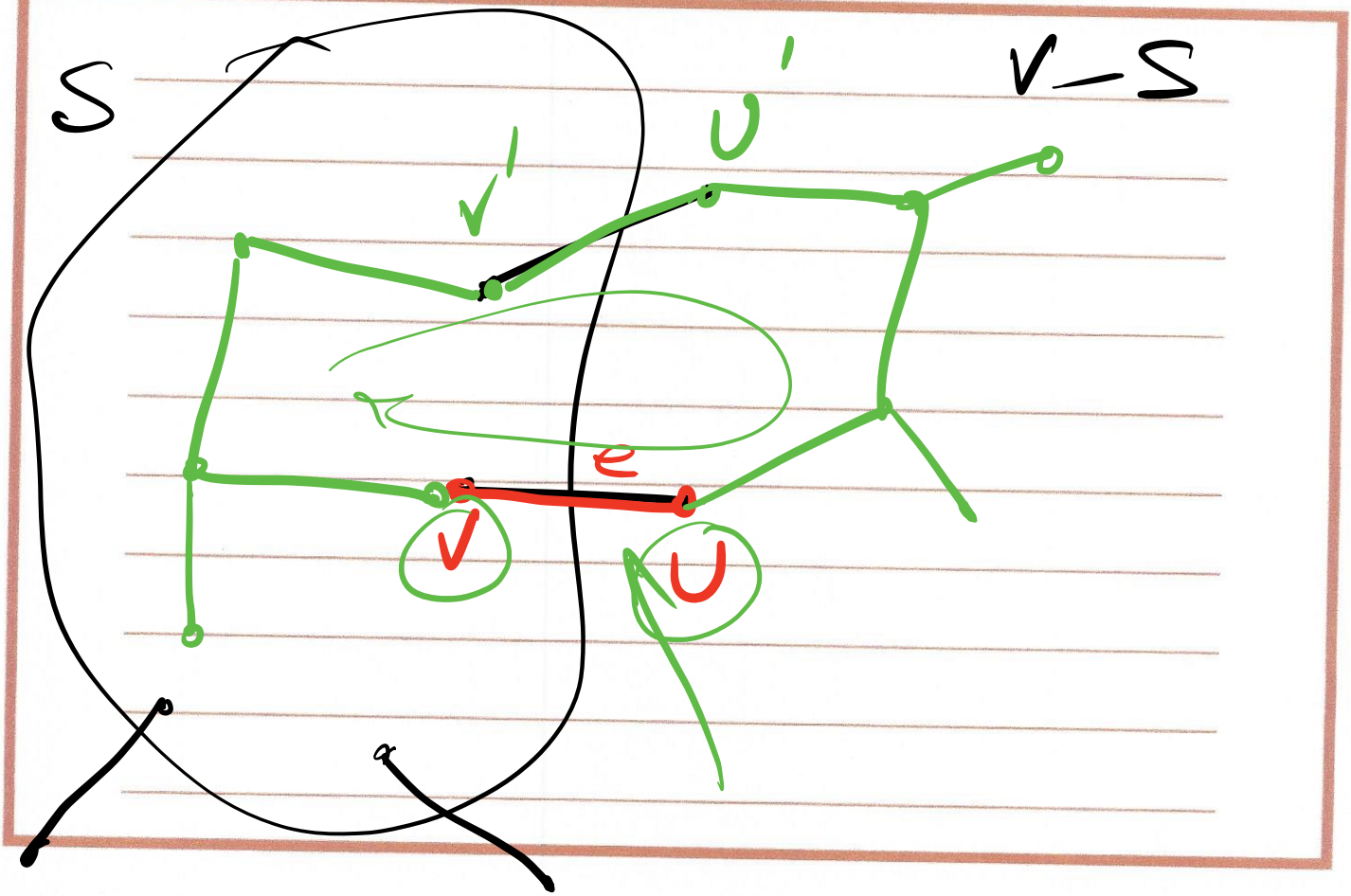


Kruskal's ——

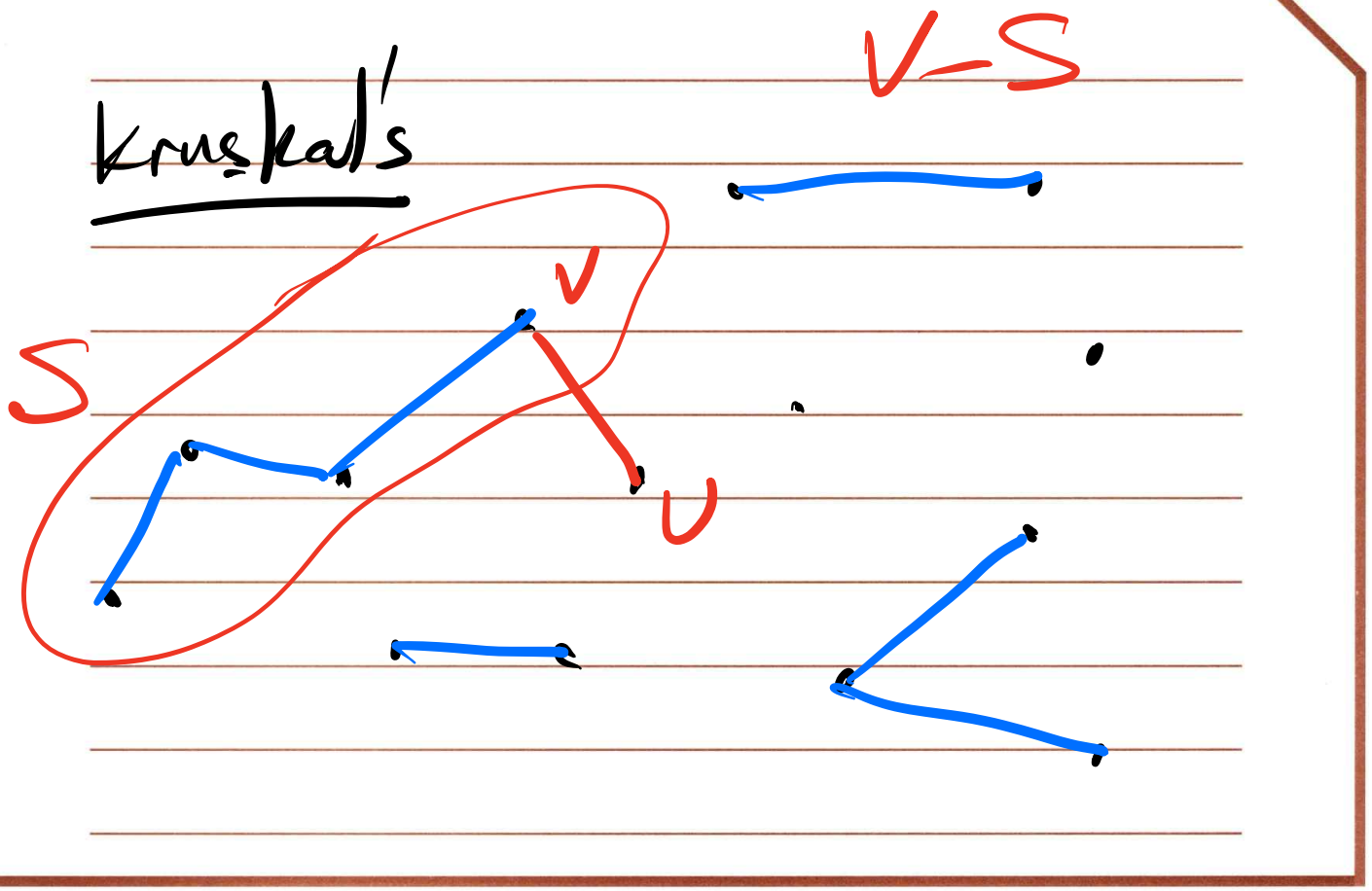
Prim's - - - -

Reverse Delete X

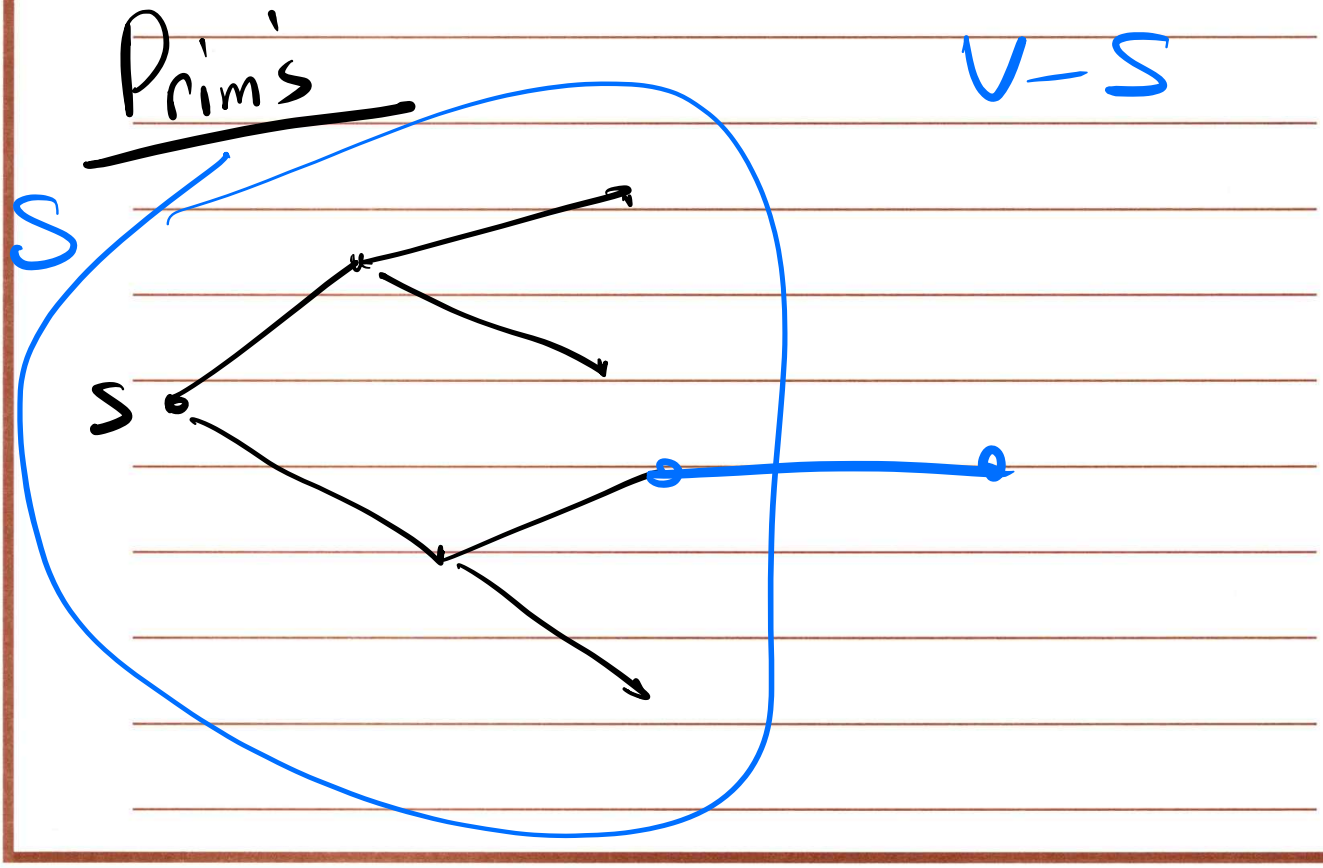
FACT: Let S be any subset of nodes that is neither empty nor equal to all of V, and let edge $e = (v, w)$ be the min cost edge with one end in S and the other end in $V - S$. Then every MST contains the edge e.



Kruskal's



Prim's



More Detailed Implementation of ~~Dijkstra's~~ Prim's

S = Null

Initialize priority Queue Q with all nodes V where d(v) is the key value.
(All d(v)'s are $=\infty$, except for s where d(s)=0)

While $S \neq V$

v = Extract_Min (Q)

$S = S \cup \{v\}$

for each vertex $u \in Adj(v)$

if d(u) > ~~d(v)~~ + le;

Decrease_key (Q, u, ~~d(v)~~ + le)

end for

end while

Kruskal's

Create an endp set for each node

A = Null

Sort edges in non-decreasing order of weight

for each edge $(u,v) \in E$, taken in this order, if u & v are NOT in the same set then

$A = A \cup \{(u,v)\}$

merge the two sets

end if

endfor

Union-Find data structure

- Make set — $O(1)$ for set size = 1
- Find set — $O(1)$ or $O(\lg n)$
- Union — $O(\lg n)$ or $O(1)$

array impl. — ptr impl.

Implementation of Kruskal's

```
A = Null
for each vertex v ∈ V
    Make-set(v)
end for
```

$O(n)$

Sort the edges of E into non-decreasing order of cost

$O(m \lg m)$

```
for each edge (u,v) ∈ E in this order.
    if Find-set(u) ≠ Find-set(v) then
        A = A ∪ {(u,v)}
        Union(u,v)
    endif
endfor
```

$O(m)$

$O(\lg n)$

overall complexity $= O(n) + O(m \lg m) + O(m \lg n)$

$= O(m \lg m)$

Prim's
$O(m \log n)$

Kruskal's
$O(m \log m)$

$O(m \log n^2)$

$O(2m \log n)$

$= O(m \log n)$

Reverse Delete

$O(m \log m)$ Sort edges — decreasing order of cost

$O(m)$ loop over edges in this order

$O(m)$ check to see if removing an edge disconnects the graph

takes $O(m^2)$

Discussion 4

1. Hardy decides to start running to work in San Francisco city to get in shape. He prefers a route that goes entirely uphill and then entirely downhill so that he could work up a sweat uphill and get a nice, cool breeze at the end of his run as he runs faster downhill. He starts running from his home and ends at his workplace. To guide his run, he prints out a detailed map of the roads between home and work with k intersections and m road segments (any existing road between two intersections). The length of each road segment and the elevations at every intersection are given. Assuming that every road segment is either fully uphill or fully downhill, give an efficient algorithm to find the shortest path (route) that meets Hardy's specifications. If no such path meets Hardy's specifications, your algorithm should determine this. Justify your answer.

2. You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node *v* to node *u* if and only if *v* is a prerequisite for *u*. Top positions are the ones which are not prerequisites for any positions. The cost of an edge *(v, u)* is the effort required to go from one position *v* to position *u*. Ivan wants to start a career and achieve a top position with minimum effort. Using the given graph can you provide an algorithm with the same run time complexity as Dijkstra's? You may assume the graph is a DAG.

3. You have a stack data type, and you need to implement a FIFO queue. The stack has the usual POP and PUSH operations, and the cost of each operation is 1. The FIFO has two operations: ENQUEUE and DEQUEUE. We can implement a FIFO queue using two stacks. What is the amortized cost of ENQUEUE and DEQUEUE operations.

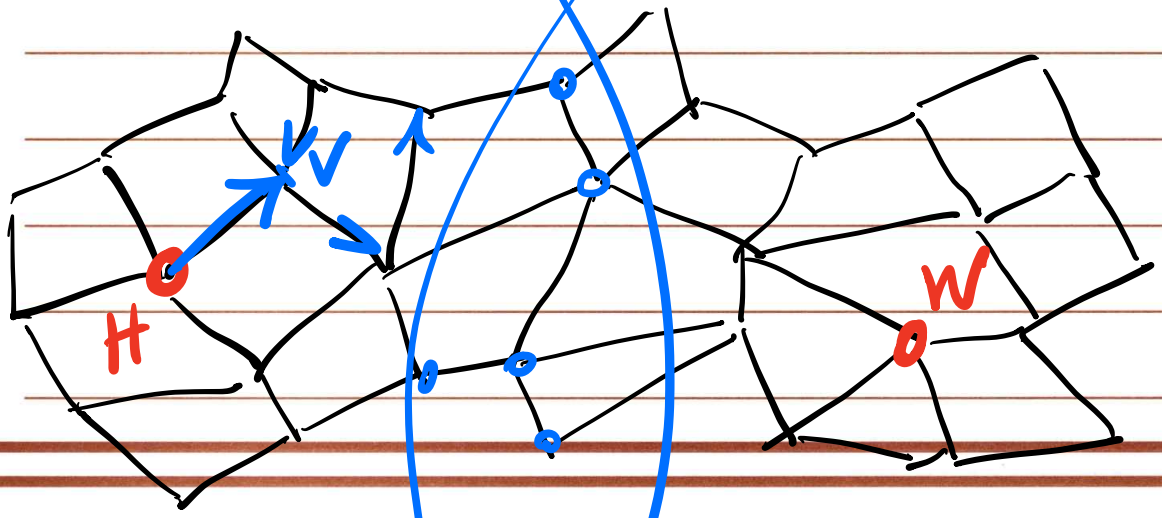
4. Given a sequence of numbers: 3, 5, 2, 8, 1, 5, 2, 7

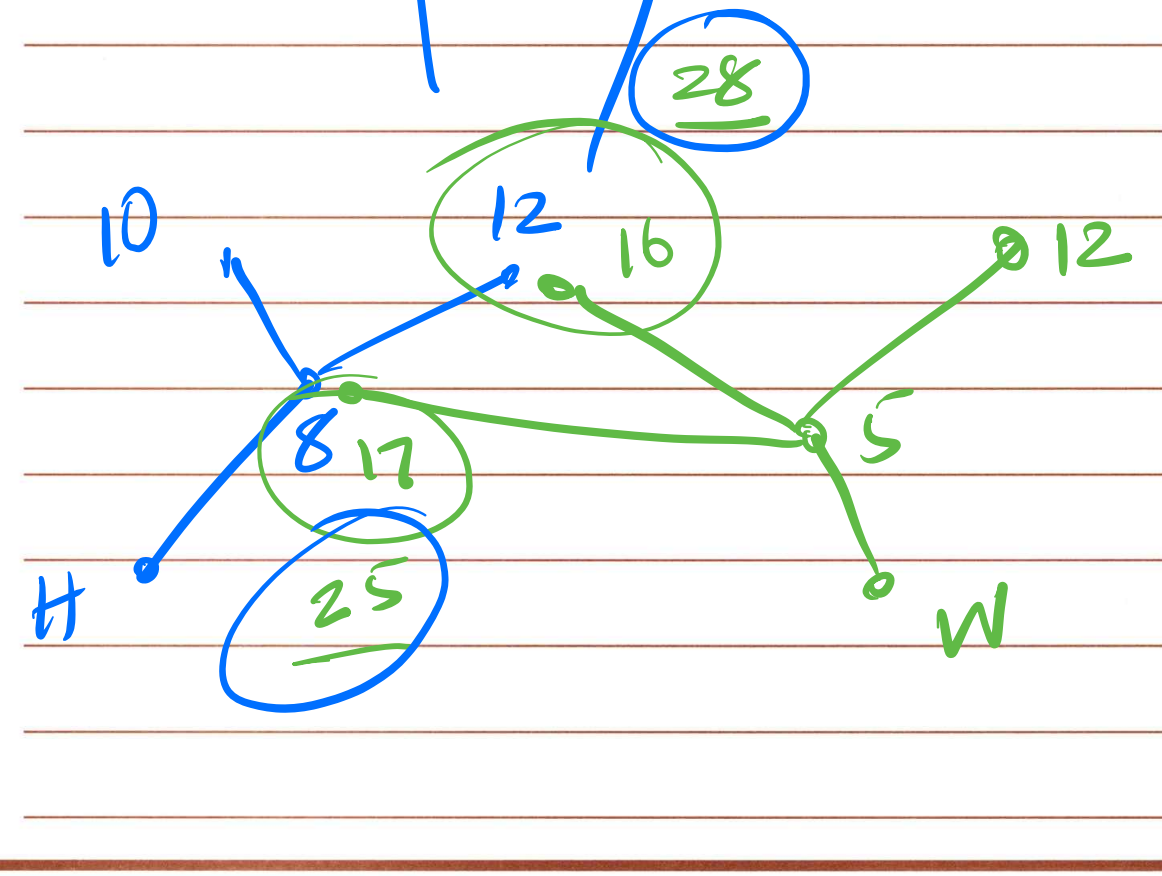
a. Draw a binomial heap by inserting the above numbers reading them from left to right
b. Show a heap that would be the result after the call to deleteMin() on this heap

5. (a): Suppose we are given an instance of the Minimum Spanning Tree problem on a graph G. Assume that all edges costs are distinct. Let T be a minimum spanning tree for this instance. Now suppose that we replace each edge cost c_e by its square, c_e^2 thereby creating a new instance of the problem with the same graph but different costs. Prove or disprove: T is still a MST for this new instance.

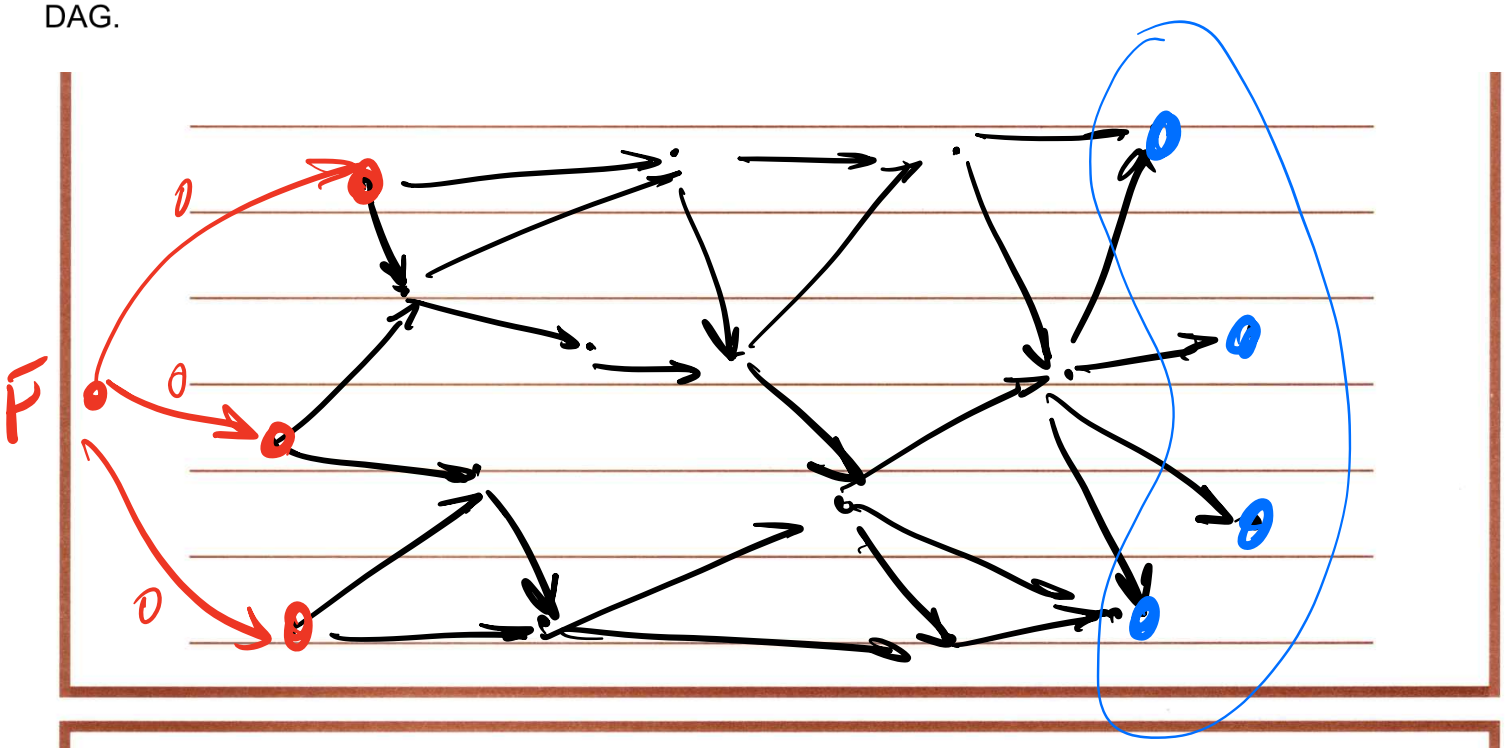
(b): Consider an undirected graph G = (V, E) with distinct nonnegative edge weights $w_e \geq 0$. Suppose that you have computed a minimum spanning tree of G. Now suppose each edge weight is increased by 1: the new weights are $c'_e = c_e + 1$. Does the minimum spanning tree change? Give an example where it changes or prove it cannot change.

1. Hardy decides to start running to work in San Francisco city to get in shape. He prefers a route that goes entirely uphill and then entirely downhill so that he could work up a sweat uphill and get a nice, cool breeze at the end of his run as he runs faster downhill. He starts running from his home and ends at his workplace. To guide his run, he prints out a detailed map of the roads between home and work with k intersections and m road segments (any existing road between two intersections). The length of each road segment and the elevations at every intersection are given. Assuming that every road segment is either fully uphill or fully downhill, give an efficient algorithm to find the shortest path (route) that meets Hardy's specifications. If no such path meets Hardy's specifications, your algorithm should determine this. Justify your answer.



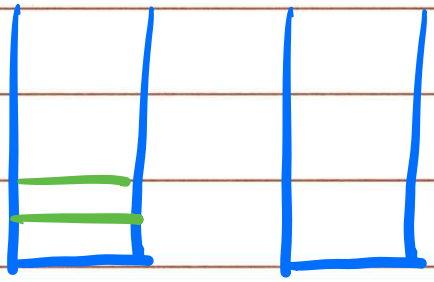


2. You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node *v* to node *u* if and only if *v* is a prerequisite for *u*. Top positions are the ones which are not prerequisites for any positions. The cost of an edge *(v, u)* is the effort required to go from one position *v* to position *u*. Ivan wants to start a career and achieve a top position with minimum effort. Using the given graph can you provide an algorithm with the same run time complexity as Dijkstra's? You may assume the graph is a DAG.



3. You have a stack data type, and you need to implement a FIFO queue. The stack has the usual POP and PUSH operations, and the cost of each operation is 1. The FIFO has two operations: ENQUEUE and DEQUEUE. We can implement a FIFO queue using two stacks. What is the amortized cost of ENQUEUE and DEQUEUE operations.

total cost of a worst case seq. of n op's



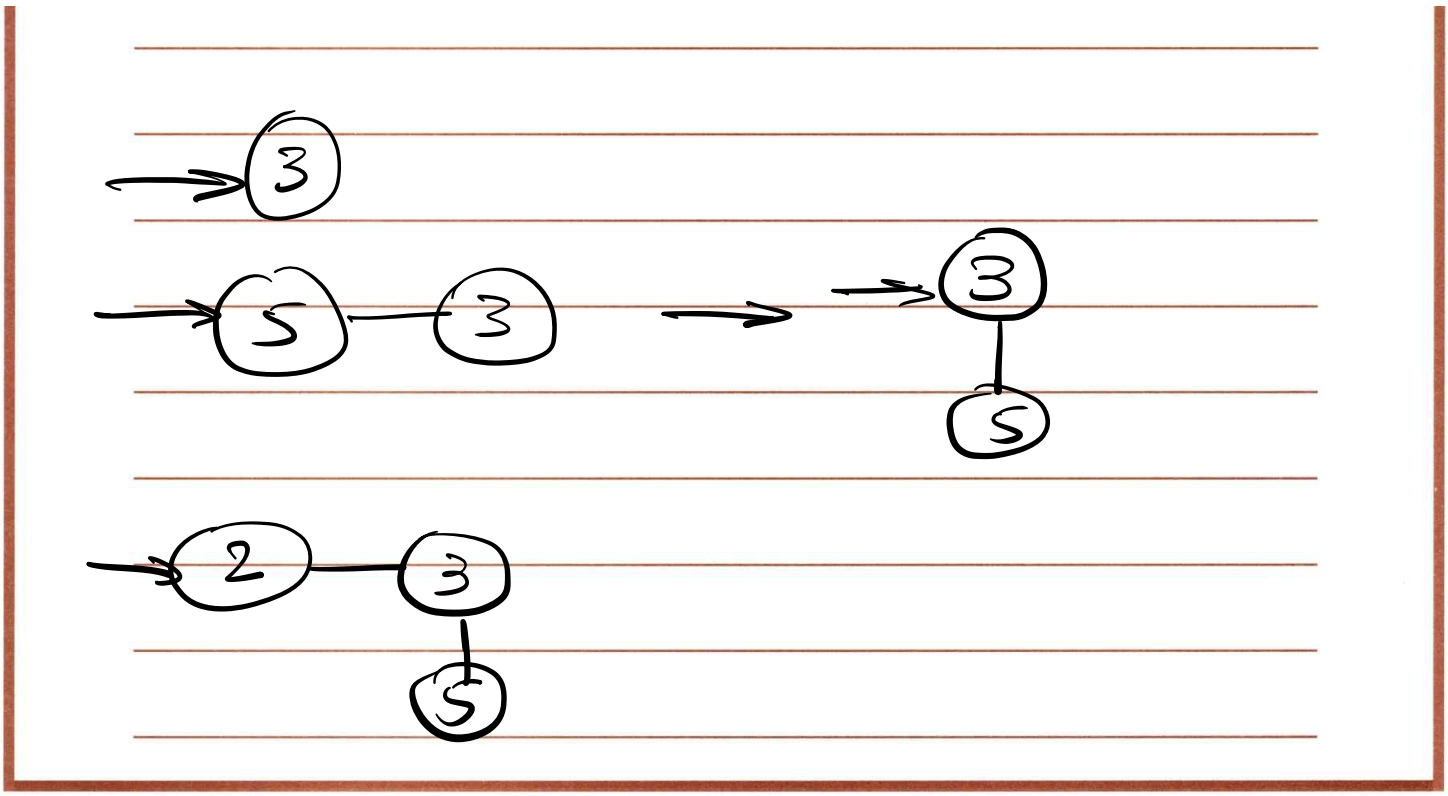
$\underbrace{n}_{\text{pushes into A}} + \underbrace{n}_{\text{pops from A}} + \underbrace{n}_{\text{pushes into B}} + 1$

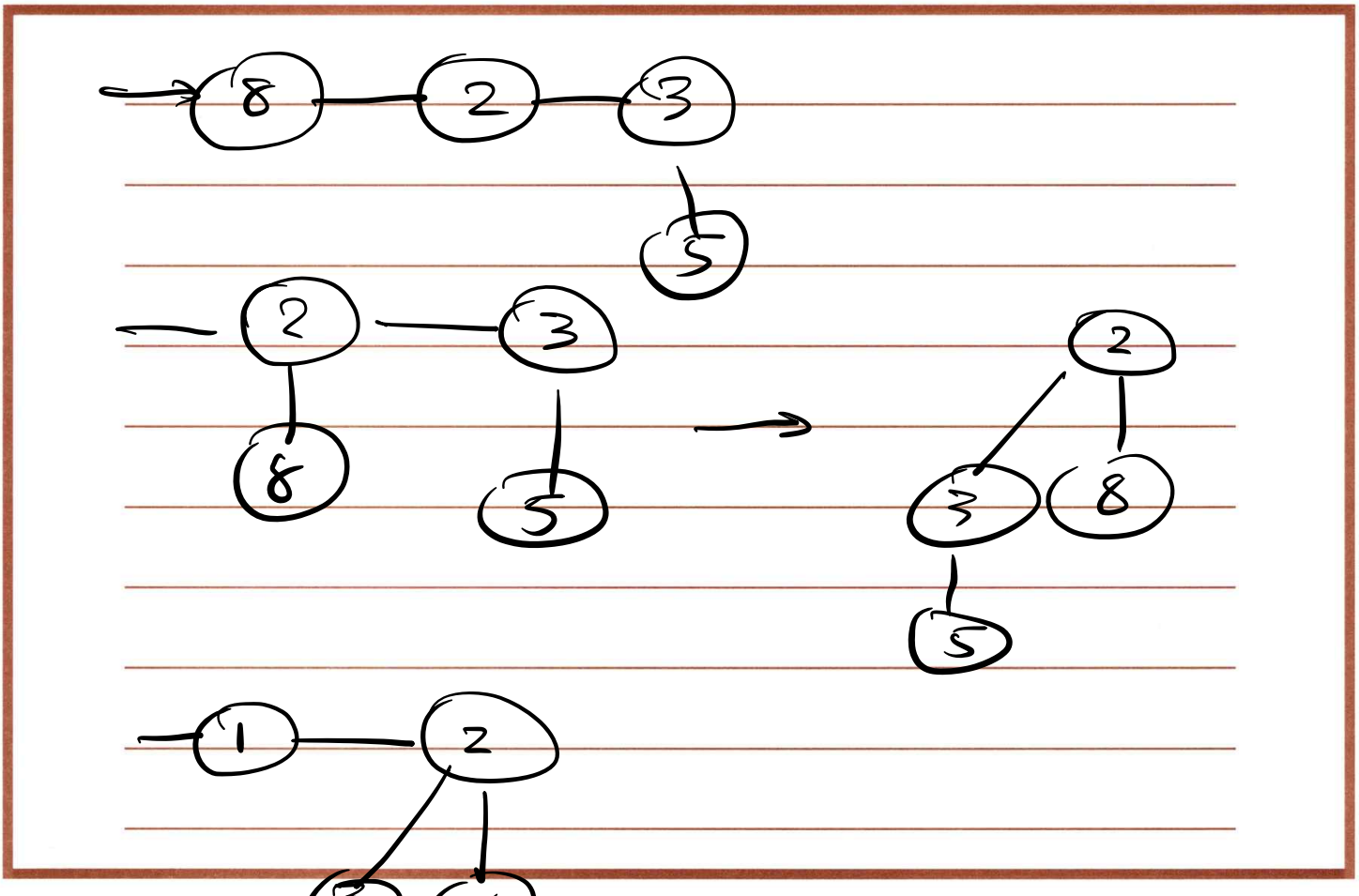
$= 3n$

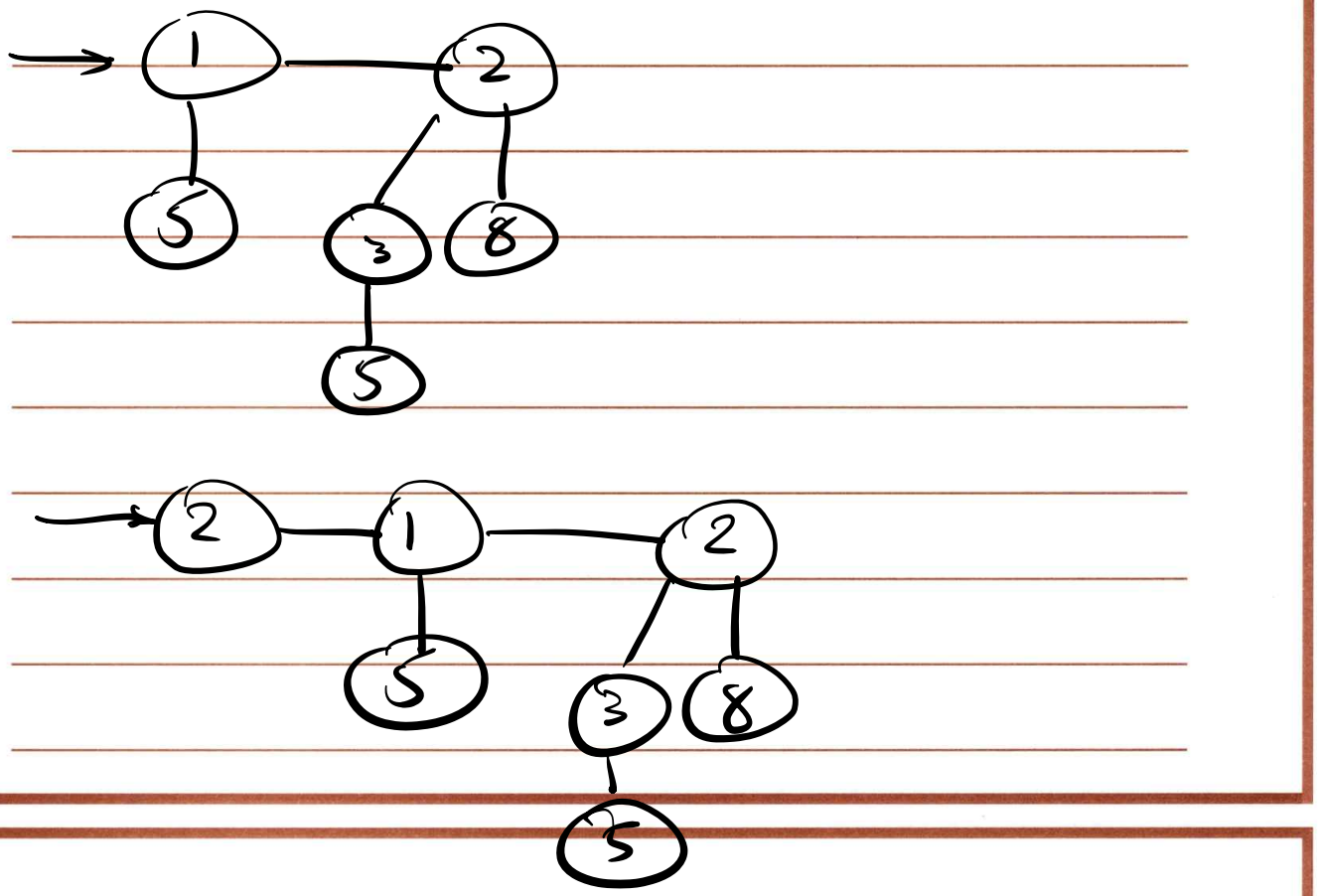
$\text{Amortized cost} = \frac{3n+1}{n} = O(1)$

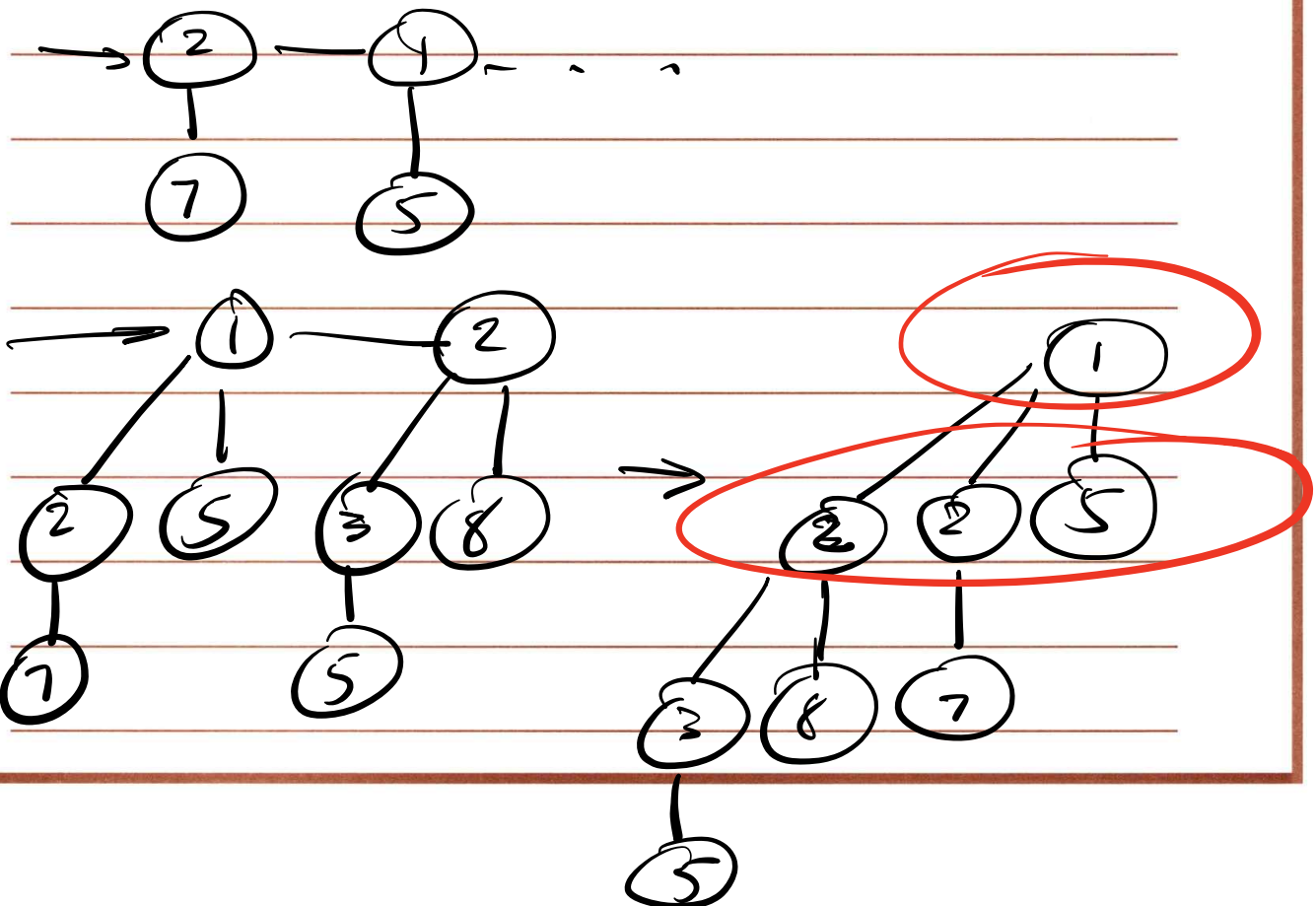
4. Given a sequence of numbers: 3, 5, 2, 8, 1, 5, 2, 7

a. Draw a binomial heap by inserting the above numbers reading them from left to right
b. Show a heap that would be the result after the call to deleteMin() on this heap






1
2
5
3
8
5
2
1
2
5
3
8
5


2
1
7
5
1
2
2
5
3
8
7
5
1
2
2
5
3
8
7
5

5. (a): Suppose we are given an instance of the Minimum Spanning Tree problem on a graph G. Assume that all edges costs are distinct. Let T be a minimum spanning tree for this instance. Now suppose that we replace each edge cost c_e by its square, c_e^2 thereby creating a new instance of the problem with the same graph but different costs. Prove or disprove: T is still a MST for this new instance.

Tree could change if we have neg. nos

MST for this new instance.

(b): Consider an undirected graph G = (V, E) with distinct nonnegative edge weights $w_e \geq 0$. Suppose that you have computed a minimum spanning tree of G. Now suppose each edge weight is increased by 1: the new weights are $c'_e = c_e + 1$. Does the minimum spanning tree change? Give an example where it changes or prove it cannot change.