CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Assignment 1 – September 11, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

answer key

Problem 1 (1+1+1+1 marks). For each question,

- subtract half a mark if the solution is not the simplest possible, e.g., if someone writes $\Theta(0.1N^3\sqrt{N})$ or $\Theta(N+1)$,
- give no mark if the growth rate class given is not correct.

(a) $T(N) = 5.3 \cdot N^3\sqrt{N} = +22N^3\log(N) + 100N\sqrt{N} = \Theta(N^3\sqrt{N})$

(b) $T(N) = 0.6\frac{\log^2(N)}{N}12.5N = \Theta(N)$

(c) $T(N) = \Theta(N \cdot \log^3(N))$

(d) $T(N) = \frac{36\log^2(N)}{N^2} + \frac{2}{3}\frac{\log(N)\cdot 3^{N+1}}{N^2} + 57 \cdot \frac{\log(N)}{N^2} = \Theta(\frac{\log(N)\cdot 3^N}{N^2})$

Problem 2 (3+3 marks). (a) $T(N) = o(f(N))$ implies $T(N) = O(f(N))$ and $T(N) \neq \Omega(f(N))$. (0.5 marks)
$f(N) = \Theta(g(N))$ implies $f(N) = O(g(N))$. (0.5 marks)
$T(N) = O(f(N))$ and $f(N) = \Theta(g(N))$ together imply $T(N) = O(g(N))$. (1 mark)
$T(N) \neq \Omega(f(N))$ and $f(N) = O(g(N))$ implies $T(N) \neq \Omega(g(N))$ and thus $T(N) = o(g(N))$. (1 mark)

(b) $T(N) = 1$, $f(N) = \log(N)$, $g(N) = N$. (1.5 marks for correct choice, 0.5 marks for every correct relation, e.g., 1 mark if $f(N) = o(g(N))$ and $T(N) = O(g(N))$, but $T(N) = \Omega(f(N))$)
Then $T(N) = O(g(N))$, because N grows faster than 1 (a constant). (0.5 marks)
$f(N) = o(g(N))$, because $\log(N) = O(N)$ and $\log(N)$ does *not* grow as fast as N. (0.5 marks)
$T(N) \neq \Omega(f(N))$, because $\log(N)$ grows faster than 1 (a constant). (0.5 marks)

Problem 3 (3+3+3 marks). For each of the following code fragments, determine the best possible asymptotic upper bound on its running time, depending on `n`. Give a brief explanation for each of your answers.

(a) The statement inside the loops has a running time of $O(1)$. (1 mark)
The inner loop iterates $O(n)$ times (namely roughly $n/2$ times), thus making the overall running time of the inner loop $O(n)$. (1 mark)
The outer loop runs n times, resulting in an overall running time of $O(n^2)$. (1 mark)

(b) The statement inside the loops has a running time of $O(1)$. (0.5 marks)
The inner loop iterates $O(n)$ times (namely roughly $n/2$ times), thus making the overall running time of the inner loop $O(n)$. (1 mark)
The outer loop runs $O(\log(n))$ times, resulting in an overall running time of $O(n \cdot \log(n))$. (1.5 marks)

(c) The running time $T(n)$ can be described (asymptotically) by $T(n) = 1$ if $n = 1$ and $T(n) = T(n-1) + 1$ if $n > 1$. (1.5 marks)
This results in $T(n) = n = O(n)$. (1.5 marks)
(Other correct explanations will also be accepted.)

Problem 4 (4+2+4 marks). (a) half marks for a reasonable approach that does not work because of a small error

(b) half marks for a reasonable approach that does not work because of a small error

(c) $n\log(n)$ is the only function for which the ratio by `result` calculated in (b) approaches a constant for growing n. Hence we may conclude that the asymptotic growth of the result of `assignment1Algorithm(n)` is $\Theta(n \cdot \log(n))$. (1.5 marks)
$n\log(n)$ is the only function for which the ratio by `timeUsed` calculated in (b) approaches a constant for growing n. Hence we may conclude that the asymptotic growth of the running time of `assignment1Algorithm(n)` is $\Theta(n \cdot \log(n))$. (1.5 marks)
$n\log(n)$ grows only slightly faster than n, whereas $n\sqrt{n}$ grows "significantly" faster than $n\log(n)$. (1 mark)

CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Assignment 2 – September 18, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

answer key

Problem 0 (1 mark). Everyone gets 1 mark, even if they don't send a joke/cartoon.

Problem 1 (6 marks). **algorithm** Let `q` be an initially empty queue. Call `levelOrder(r)`.

```
levelOrder(n)
  q.enqueue(n)
  while not q.empty do
    node := q.dequeue()
    output(node->element)
    if (node->left != NULL)
      q.enqueue(node->left)
    if (node->right != NULL)
      q.enqueue(node->right)
```

(4 marks – give partial marks for reasonable but incomplete approaches)

Since every node is enqueued exactly once and in every iteration of the while loop one node is dequeued, the while loop runs through $N = O(N)$ iterations. Every iteration consumes a constant amount of time ($O(1)$). Hence the running time of the algorithm is $O(N \cdot 1) = O(N)$. (2 marks – subtract one mark if the argument is incomplete but at least partly reasonable)

Problem 2 (6+2 marks). see hand-written sheets

Problem 3 (3+4 marks).

(a) The running time depends only on the length of the array. Therefore the worst case, average case and best case are all equal. [1 mark]
The time to insert is $\Theta(N)$. [1 mark]
This is because the N elements in the array all have to be moved by one slot. [1 mark]

(b) The cost of the sequence of insertions is $1+2+3+\ldots+N$ units, because for the i^{th} insertion $i-1$ entries first have to be moved. [1 mark]
This equals $\frac{N(N+1)}{2} = O(N^2)$. [1 mark]
This means that the amortized running time over N insertions is $O(N)$. [2 marks]
(Actually, here no real amortization happens, because worst case cost equals amortized cost.)

Problem 4 (2+2+4 marks). (a) The worst case running time of a single execution is $\Theta(N)$. [1 mark]
This happens when the array contains only `1`s. [1 mark]

(b)

`a[5]`	`a[4]`	`a[3]`	`a[2]`	`a[1]`	`a[0]`
0	0	0	0	0	0
0	0	0	0	0	**1**
0	0	0	0	**1**	**0**
0	0	0	0	1	**1**
0	0	0	**1**	**0**	**0**
0	0	0	1	0	**1**
0	0	0	1	**1**	**0**
0	0	0	1	1	**1**
0	0	**1**	**0**	**0**	**0**
0	0	1	0	0	**1**
0	0	1	0	**1**	**0**
0	0	1	0	1	**1**
0	0	1	**1**	**0**	**0**
0	0	1	1	0	**1**
0	0	1	1	**1**	**0**
0	0	1	1	1	**1**
0	**1**	**0**	**0**	**0**	**0**

Subtract 1 mark if it is not correctly marked which bits are flipped. Subtract 1 mark if there is more than only a tiny mistake in the table.

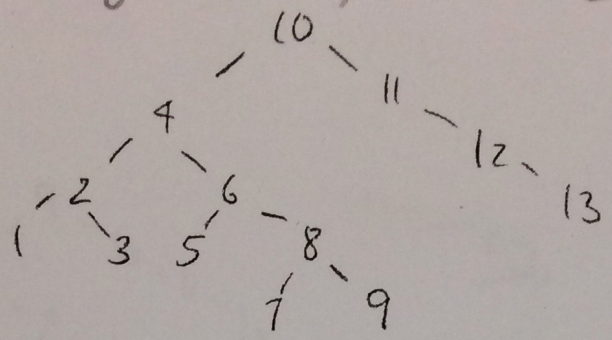
(c) It is obvious in (b) that in every step the bit in `a[0]` is flipped, in every second step the bit in `a[1]` is flipped, in every fourth step, the bit in `a[2]` is flipped, and so on. Therefore, in N operations, `a[0]` flips N times, `a[1]` flips $\lfloor N/2 \rfloor$ times, and so on, i.e., `a[i]` flips $\lfloor N/2^i \rfloor$. [1 mark]
The total number of flips is therefore

$$\sum_{i=0}^{k-1} \lfloor \frac{N}{2^i} \rfloor < N \sum_{i=0}^{k-1} \frac{1}{2^i} = 2N\,.$$

[1 mark]
The worst case time for a sequence of N operations starting from the empty array is therefore $O(N)$. [1 mark]
The amortized cost is therefore $O(1)$. [1 mark]

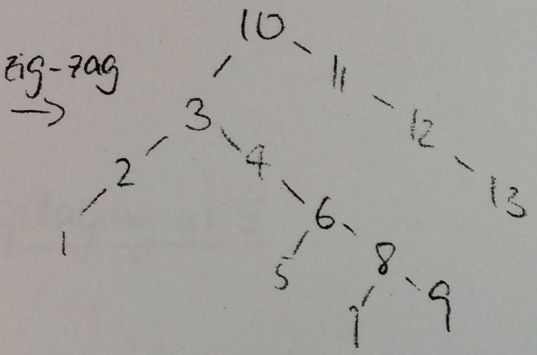
assignment 2, Problem 2

ial tree:

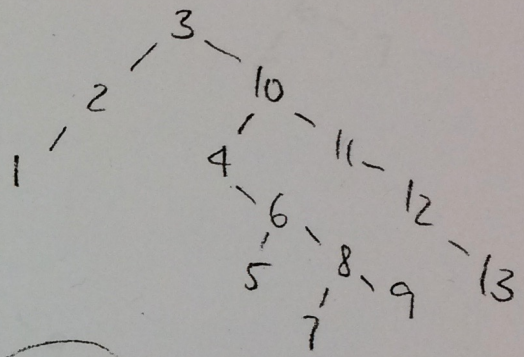


splaying at 3:

zig-zag →



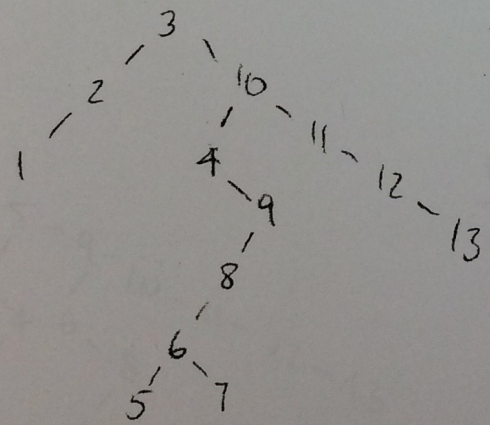
zig →



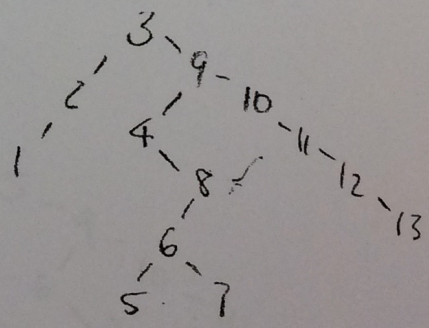
(1 mark)

splaying at 9:

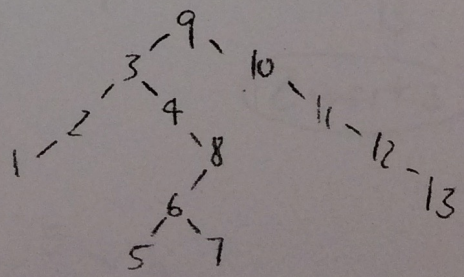
zig-zig →



zig-zag →



zig →

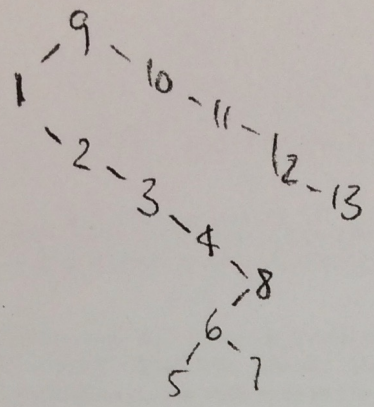


(2 marks)

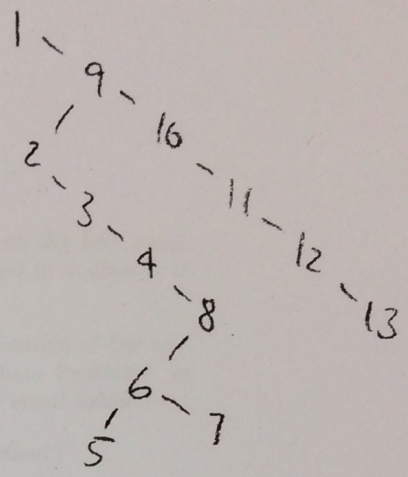
...

~~Splaying~~ at 1

zig-zig →

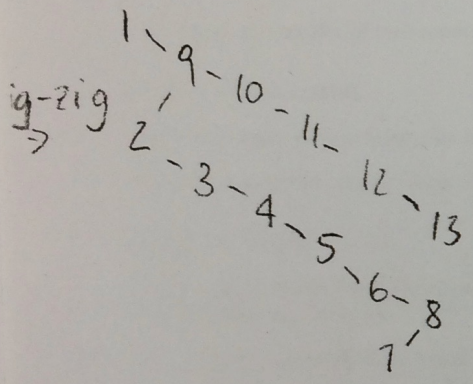


zig →

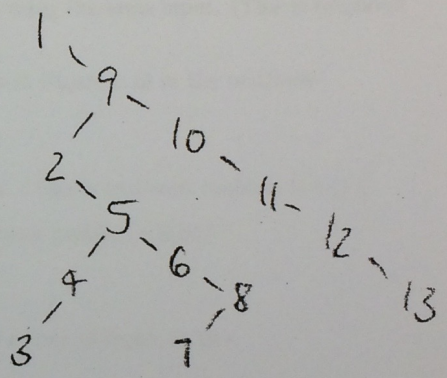


(1 mark)

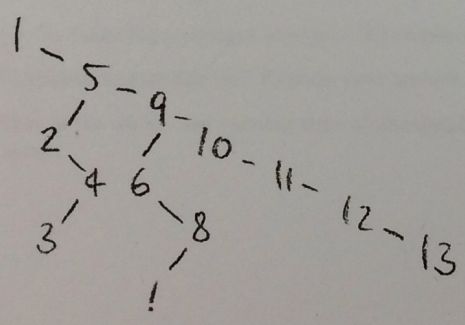
Splaying at 5



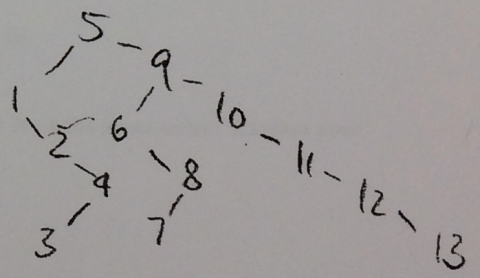
zig-zig →



zig-zag →



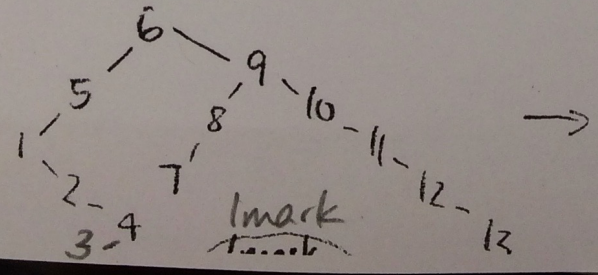
zig →



(2 marks)

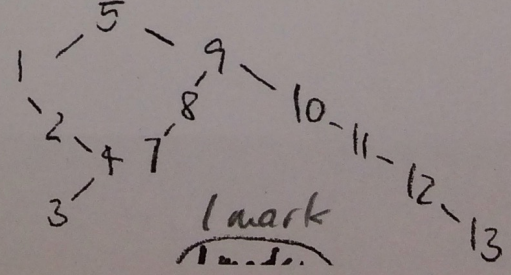
c) delete 6 : first splay at 6, then remove

zig-zag →



1 mark

→



1 mark

CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Assignment 3 – September 25, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

answer key

Problem 1 (3+3+1 marks). see hand-written sheets

Problem 2 (2+3 marks).

(a) Proof by contradiction: Assume the max. item k is not in a leaf, but in an inner node n. Then n has a child whose key is smaller than k. This violates the heap-order property. [partial marks if some reasoning makes sense.]

(b) Proof by induction on N.

induction base: $N = 1$. The heap consists only of one node, which is a leaf. Since $\lceil \frac{N}{2} \rceil = 1$, there are exactly $\lceil \frac{N}{2} \rceil$ many leaves. [1 mark]

inductive hyposthesis: Suppose any binary heap with $N \geq 1$ nodes has exactly $\lceil \frac{N}{2} \rceil$ leaves. [0.5 marks]

inductive step: $N \rightsquigarrow N+1$. Consider a binary heap B with $N+1$ nodes.

If $N+1$ is even then removing a node does not change the number of leaves. By inductive hypothesis, the heap B has $\lceil \frac{N}{2} \rceil$ leaves. Since N is odd, this number equals $\lceil \frac{N+1}{2} \rceil$.

If $N+1$ is odd then removing a node decreases the number of leaves by one. By inductive hypothesis, the heap B has $\lceil \frac{N}{2} \rceil + 1$ leaves. Since N is even, this number equals $\lceil \frac{N+1}{2} \rceil$.

[1.5 marks; give partial marks if students forget to do the case distinction]

Problem 3 (4+2+2 marks).

(a) [give partial marks for partially correct code: 1 for buildHeap, 1 for percolateDown, 2 for rest]

(b) It sorts the given array in decreasing order. [1 mark]

To see this, note that after the heap is built, the smallest entry sits in the root (array position 0). This smallest element is then swapped to the end of the array and never touched again. After the swap, the heap order property is re-established by percolating down. Then again the smallest of the remaining entries is in position 0 of the array and the process is repeated. [1 mark]

(c) The worst-case running time is $\Theta(N \log(N))$. [1 mark; give this mark also if the students just write $O(N \log(N))$.]

Building the heap has a worst case running time of $\Theta(N)$. This is the worst case cost of step 1.

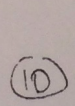
Step 2 executes a loop N times: for each value of j starting at $j = N-1$ and decrementing down to $j = 1$, the inner part of the loop percolates down in an array of j elements, at a cost of $\Theta(\log(j))$. This results in a cost of

$$\sum_{j=1}^{N-1} \log(j) = \log((N-1)!) = O(N \log(N)) \,.$$

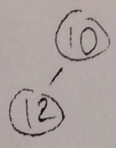
[1 mark for explanation of $\Theta(N)$ or of $O(N)$.]

a) 10, 12, 1, 14, 6, 8, 8, 15, 7, 9, 7, 4, 1, 13, 2

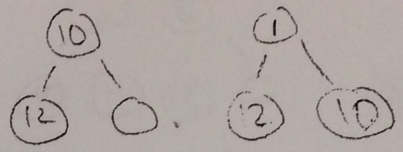
10:



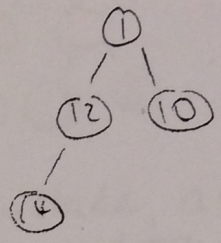
12:



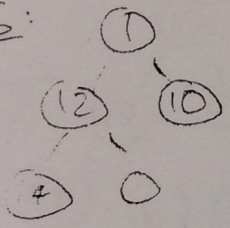
1:

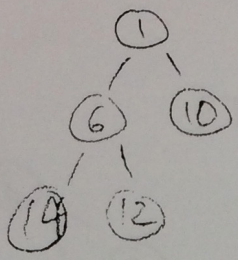


14

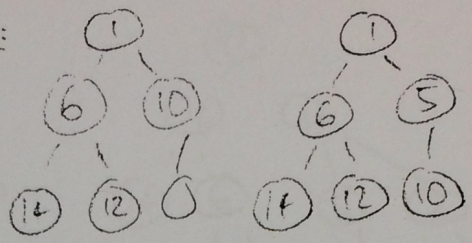


6:

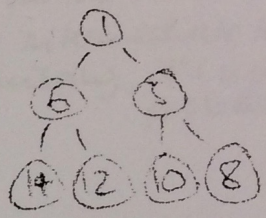




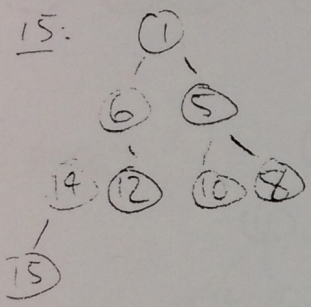
5:



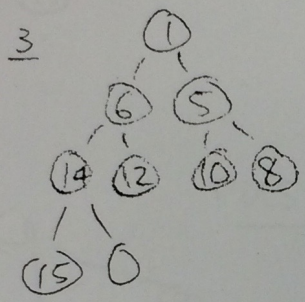
8:

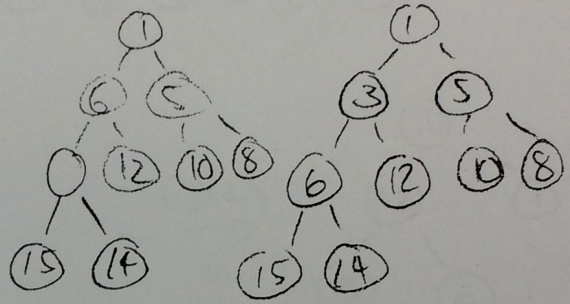


15:

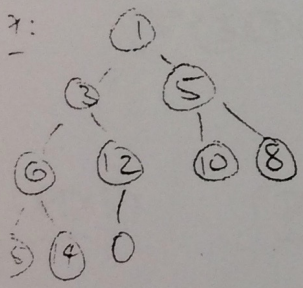


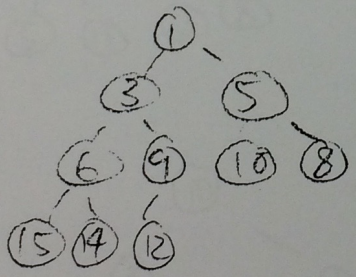
3



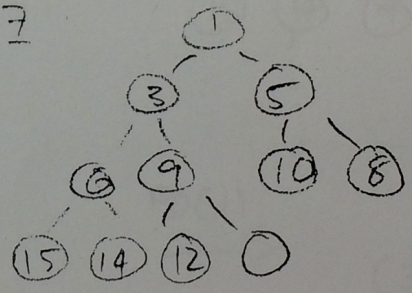


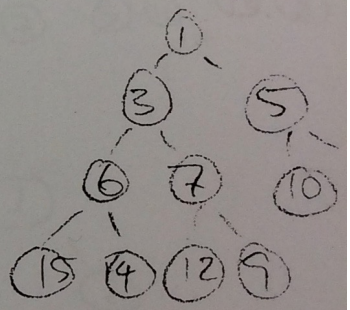
7:

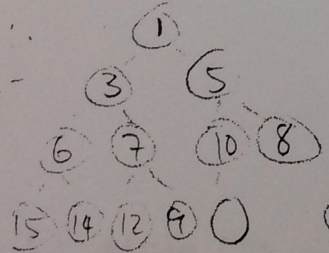


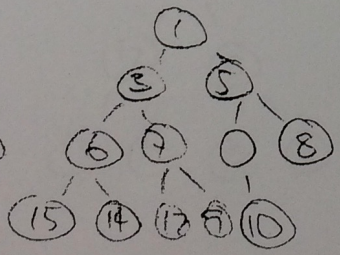


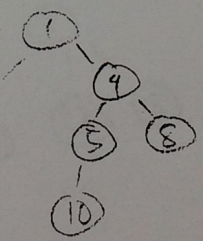
7



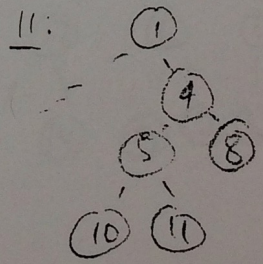




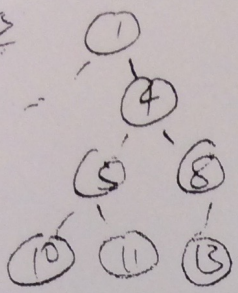




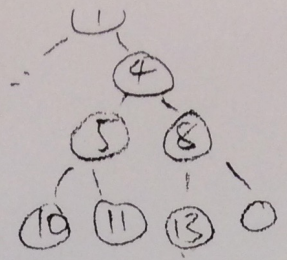
11:

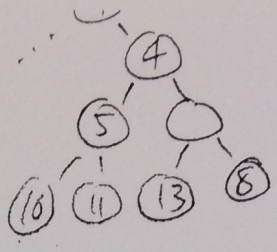


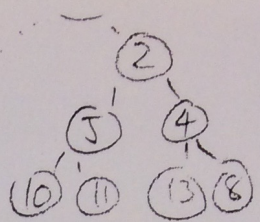
13



2.

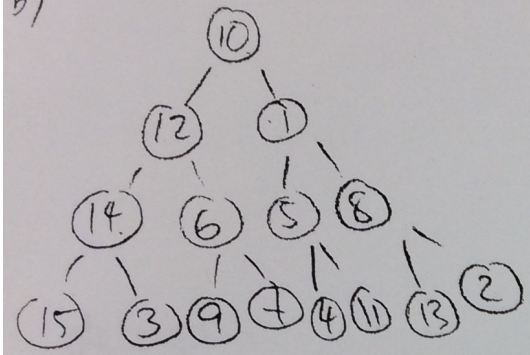




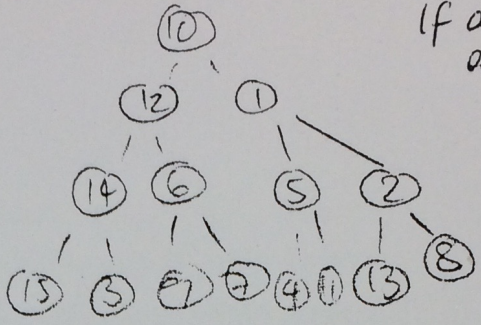


subtract 1 mark for first mistake, 0.5 marks for every further mistake.
If a few intermediate trees are missing, it's no problem

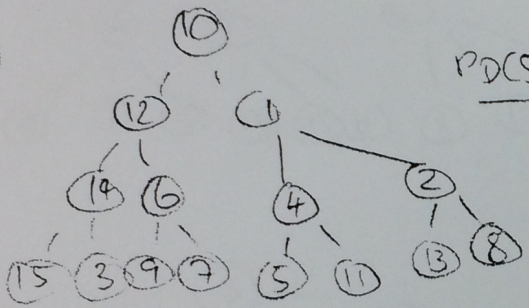
b)



PD(7) →

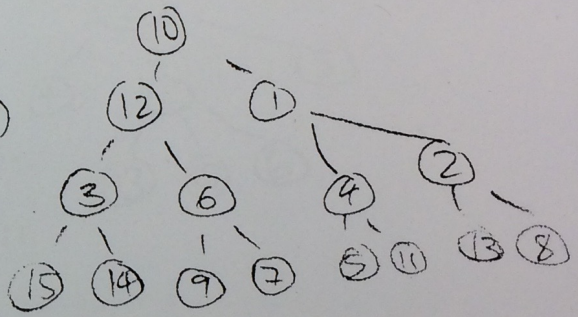


PD(6) →



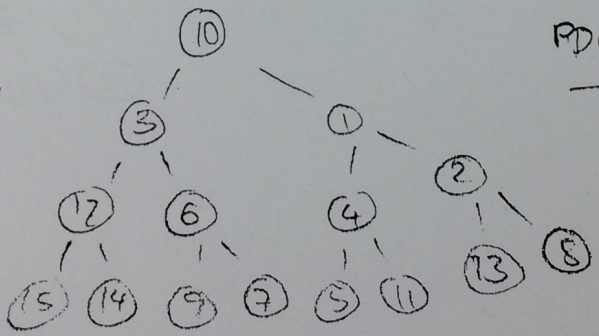
PD(5) →

PD(4) →

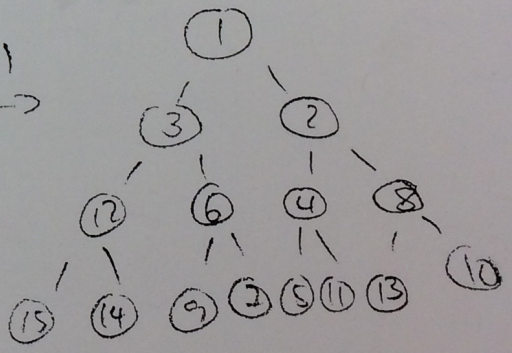


PD(3) →

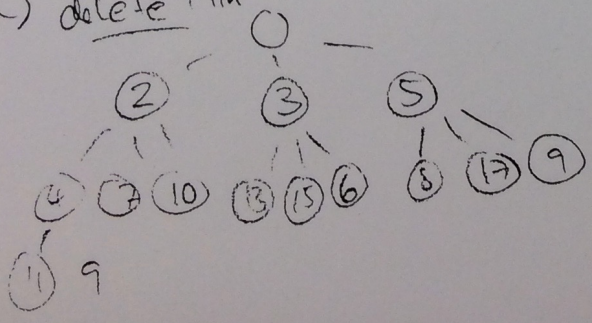
PD(2) →

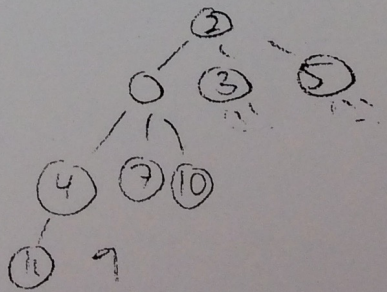


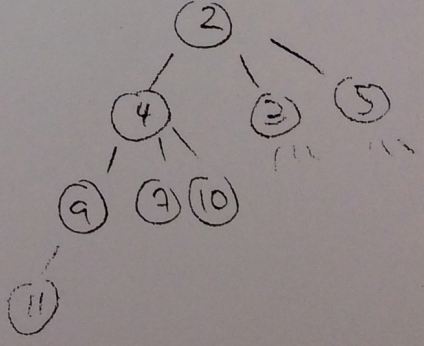
PD(1) →



c) delete Min

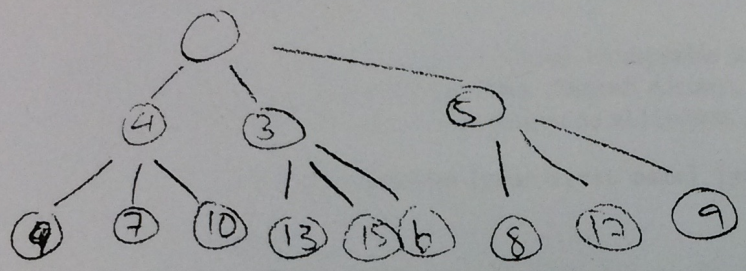


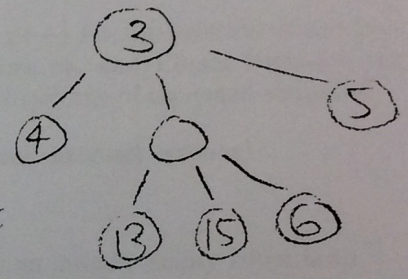


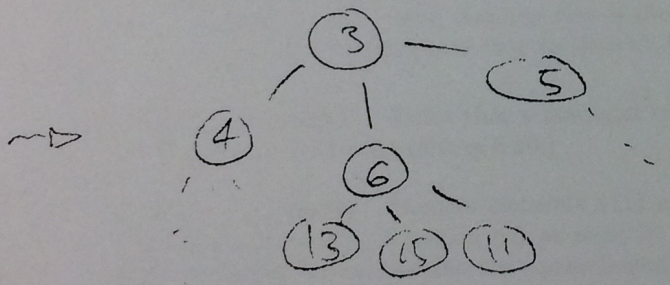


0.5 marks

deleteMin







0.5 marks

CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Assignment 4 – October 2, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

answer key

Problem 1 (2+4 marks).

(a) see hand-written sheets

(b) Placing all single-node heaps on a queue takes $O(N)$ time. [0.5 marks]

In the first step, $N/2$ pairs of heaps of size 1 are dequeued (for $O(1)$ per pair) and merged (for $O(\log(2))$ per pair). In the second step, $N/4$ pairs of heaps of size 2 are dequeued (for $O(1)$ per pair) and merged (for $O(\log(4))$ per pair). In the third step, $N/8$ pairs of heaps of size 4 are dequeued (for $O(1)$ per pair) and merged (for $O(\log(8))$ per pair). ... Finally, in the last step, $N/N = 1$ pair of heaps of size $N/2$ is dequeued (for $O(1)$) and then merged (for $O(\log(N))$). [2 marks]

In total, we get a worst case runtime of

$$O(N/2\log(2) + N/4\log(4) + \cdots + N/N\log(N)) = O(N\sum_{i=1}^{\log(N)} \frac{1}{2^i}\log(2^i)) = O(N\sum_{i=1}^{\log(N)} \frac{i}{2^i}) \text{ which is in } O(N)\,.$$

[1.5 marks]

Problem 2 (4 marks).

Proof by induction on k.

induction base: $k = 1$. In any binomial tree in B_1, the root has only one child, which consists of a single node. This child is a binomial tree in B_0. [1 mark]

inductive hypothesis: Suppose that, for some fixed k, in any binomial tree in B_k, the root has trees from B_0, B_1, ..., B_{k-1} as children. [0.5 marks]

inductive step: $k \rightsquigarrow k+1$. Consider a binomial tree $\mathcal{T}$ in B_{k+1}.

By definition, such a tree consists of a binomial tree $\mathcal{T}_1$ from B_k, to the root of which we append a binomial tree $\mathcal{T}_2$ from B_k. [1 mark]

The children of $\mathcal{T}$ are therefore $\mathcal{T}_2$ (which is a tree from B_k) and all the children of $\mathcal{T}_1$ (which, by inductive hypothesis, are trees from B_0, B_1, ... B_{k-1}). [1 mark]

Therefore, $\mathcal{T}$ has trees from B_0, B_1, ..., B_k as children. [0.5 marks]

Problem 3 (5 marks). Since `a[i]` and `a[i+k]` were in the wrong order initially, the pair $(i, i+k)$ is an inversion. If we exchange these two elements, they will be in the correct order. Hence at least one inversion is removed when swapping `a[i]` and `a[i+k]`. [0.5 marks]

Since we swap only `a[i]` and `a[i+k]`, each inversion removed by that swap must contain the index i or the index $i+k$ (or both). [1 mark]

The elements in `a[0..i-1]` cannot be involved in any removed inversion, because their relative position to `a[i]` and `a[i+k]` is not changed by the swap. The same holds for the elements in `a[i+k+1..MAX]`. [1 mark]

Hence every removed inversion, except for the pair $(i, i+k)$, contains exactly one of the indices i and $i+k$ and exactly one of the indices in $\{i+1, \ldots, i+k-1\}$. Since the latter set has $k-1$ elements, we can form $2(k-1) = 2k-2$ such inversions. Adding the one inversion $(i, i+k)$ gives the desired upper bound of $2k-1$ on the number of removed inversions. [1.5 marks]

An array meeting the lower bound would for instance have the elements 1, 2, 3, ..., k in positions i, $i+1$, $i+2$, ..., $i+k-1$, and then the element 0 in position $i+k$. The relevant inversions before swapping are

$$(i, i+k),\ (i+1, i+k),\ (i+2, i+k),\ \ldots, (i+k-1, i+k)\,.$$

After the swap, $(i, i+k)$ is no longer an inversion, but the pairs

$$(i+1, i+k),\ (i+2, i+k),\ \ldots, (i+k-1, i+k)$$

remain inversions. [0.5 marks]

An array meeting the upper bound would for instance have the elements $k+1,\ 2,\ 3,\ \ldots,\ k,\ 1$ in positions $i,\ i+1,\ i+2,\ \ldots,\ i+k$. The relevant inversions before swapping are

$$(i, i+1),\ (i, i+2),\ (i, i+3),\ \ldots, (i, i+k)$$

as well as

$$(i+1, i+k),\ (i+2, i+k),\ \ldots, (i+k-1, i+k)$$

for a total of $2k-1$ inversions. After the swap, we get the elements in `a[i..i+k]` all in order, so none of these inversions remain. [0.5 marks]

Problem 4 (4+5 marks).

(a) see hand-written sheets

(b) [give partial marks for partially correct code: 2 marks for the first correct sorting algorithm with the first version of a gap sequence, 1 more mark for the version with the seond gap sequence, and 1 more mark for the version with the third gap sequence. 1 mark for the input/output examples including number of comparisons.]

CS 340 20131O Assignment 4

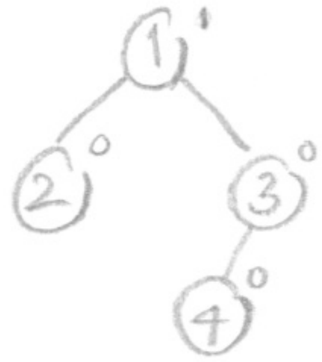
Problem 1(a) [2 marks]

insert 1:

insert 2:

insert 3:

insert 4:



insert 5:

insert 6:

insert 7:

[subtract 1 mark for each mistake]

Problem 4(a) gap sequence 7, 3, 1 [0.5 marks]

										# comparisons [1.5 marks]
g=7	77	17	66	19	30	24	64	14	23	1
	14	17	66	19	30	24	64	77	23	1
g=3	14	17	66	19	30	24	64	77	23	1
	14	17	66	19	30	24	64	77	23	1
	14	17	66	19	30	24	64	77	23	1
	14	17	24	19	30	66	64	77	23	1
	14	17	24	19	30	66	64	77	23	1
	14	17	24	19	30	66	64	77	23	2
	14	17	24	19	30	66	64	77	23	
g=1	14	17	23	19	30	24	64	77	66	1
	14	17	23	19	30	24	64	77	66	1
	14	17	23	19	30	24	64	77	66	2
	14	17	19	23	30	24	64	77	66	1
	14	17	19	23	30	24	64	77	66	2
	14	17	19	23	24	30	64	77	66	1
	14	17	19	23	24	30	64	77	66	1
	14	17	19	23	24	30	64	77	66	2
	14	17	19	23	24	30	64	66	77	

[2 marks]

total: 20 comparisons

Assignment 5

Problem 1

(a)

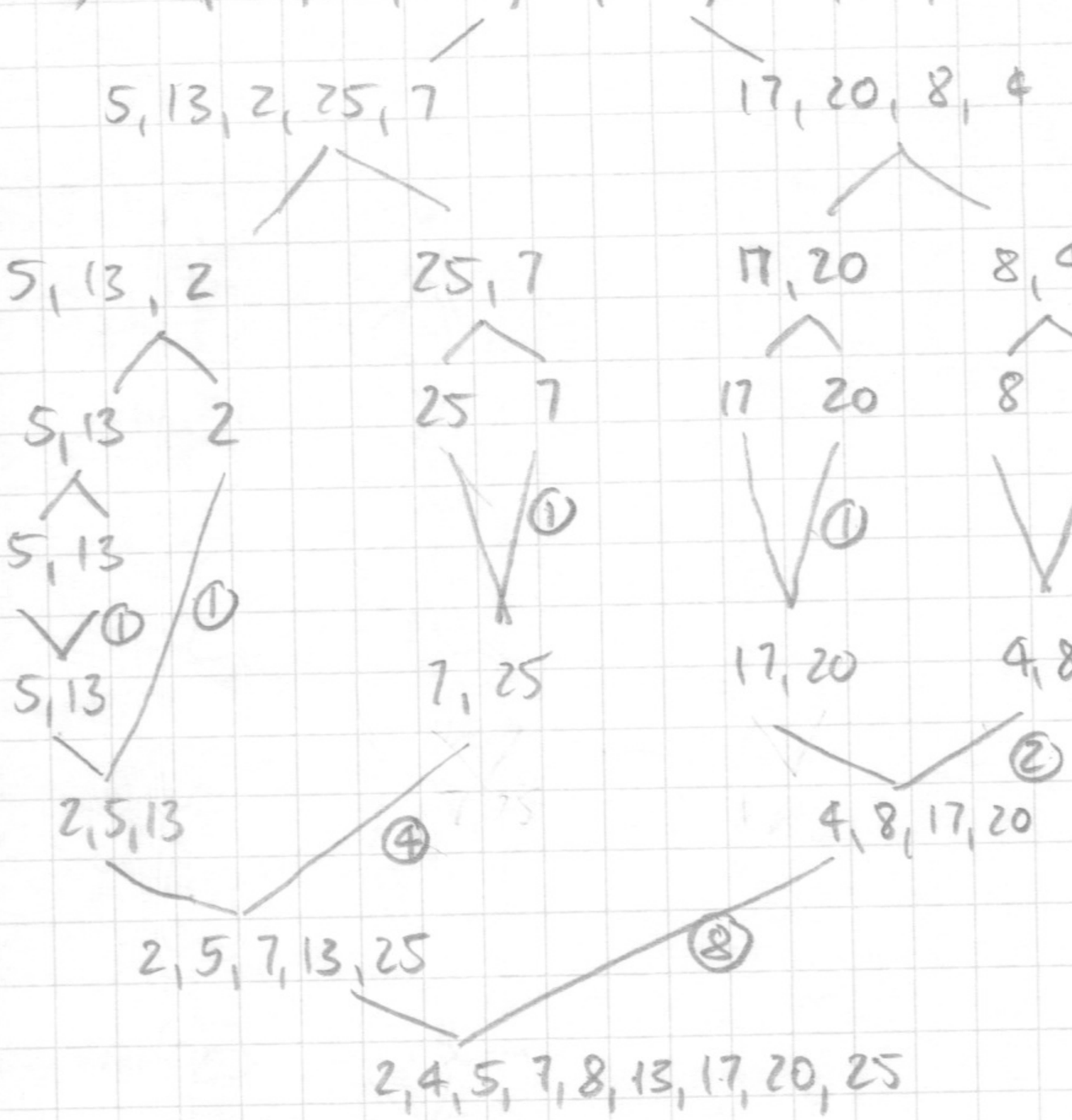
	gap	comp.
5, 13, 2, 25, 7, 17, 20, 8, 4	7	1
5, 13, 2, 25, 7, 17, 20, 8, 4		1
5, 4, 2, 25, 7, 17, 20, 8, 13	3	1
5, 4, 2, 25, 7, 17, 20, 8, 13		1
5, 4, 2, 25, 7, 17, 20, 8, 13		1
5, 4, 2, 25, 7, 17, 20, 8, 13		2
5, 4, 2, 20, 7, 17, 25, 8, 13		1
5, 4, 2, 20, 7, 17, 25, 8, 13		2
5, 4, 2, 20, 7, 13, 25, 8, 17	1	1
4, 5, 2, 20, 7, 13, 25, 8, 17		2
2, 4, 5, 20, 7, 13, 25, 8, 17		1
2, 4, 5, 20, 7, 13, 25, 8, 17		2
2, 4, 5, 7, 20, 13, 25, 8, 17		2
2, 4, 5, 7, 20, 13, 25, 8, 17		1
2, 4, 5, 7, 13, 20, 25, 8, 17		4
2, 4, 5, 7, 8, 13, 20, 25, 17		3
→ 2, 4, 5, 7, 8, 13, 17, 20, 25		

total 26 comparisons

2 marks for correct sorting
2 marks -"- counting comparisons

subtract 0.5 marks per mistake

(6) 5, 13, 2, 25, 7, 17, 20, 8, 4

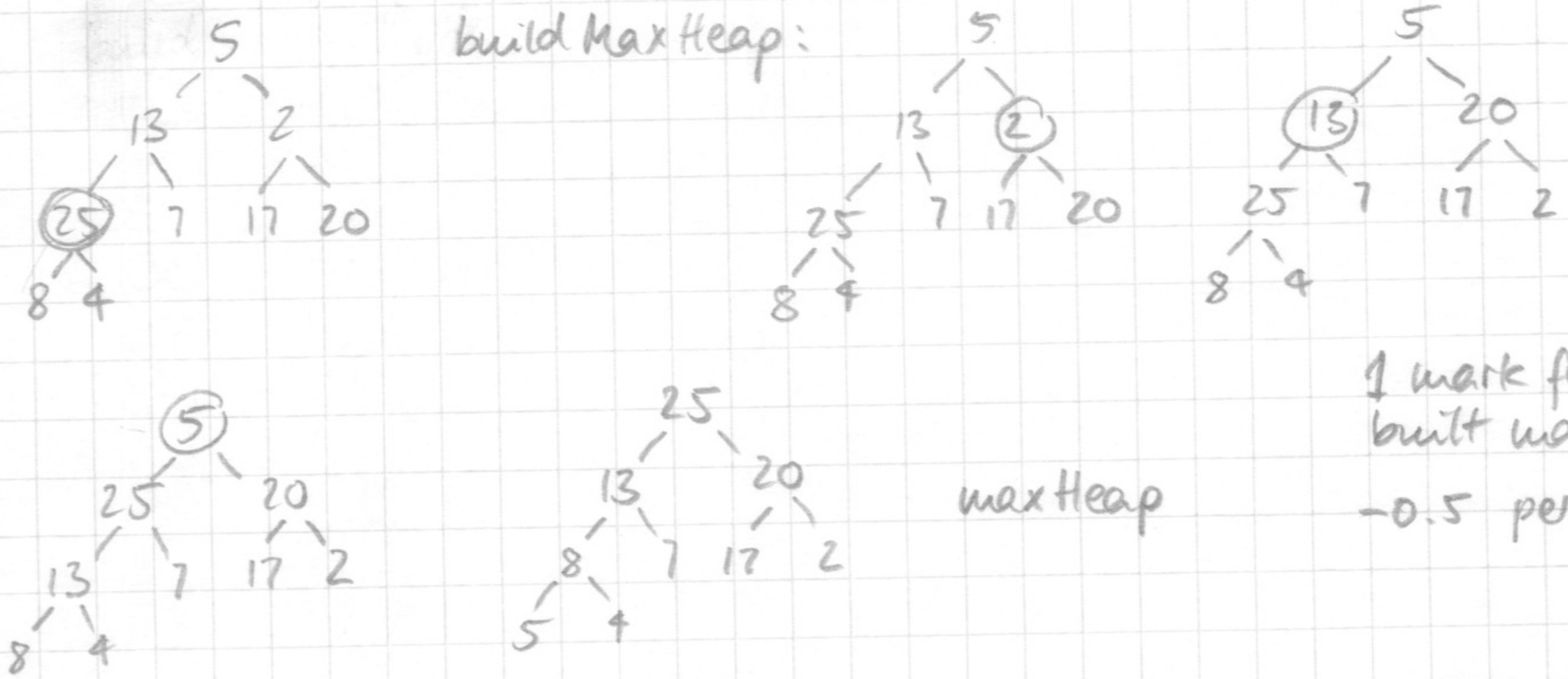


same marking rule as for (a)

total 19 comparisons

(c) 5, 13, 2, 25, 7, 17, 20, 8, 4

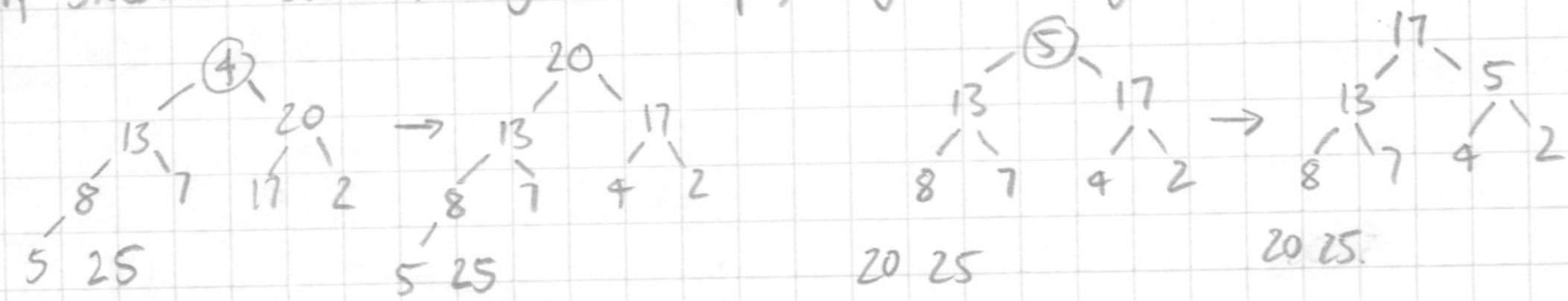
build MaxHeap:

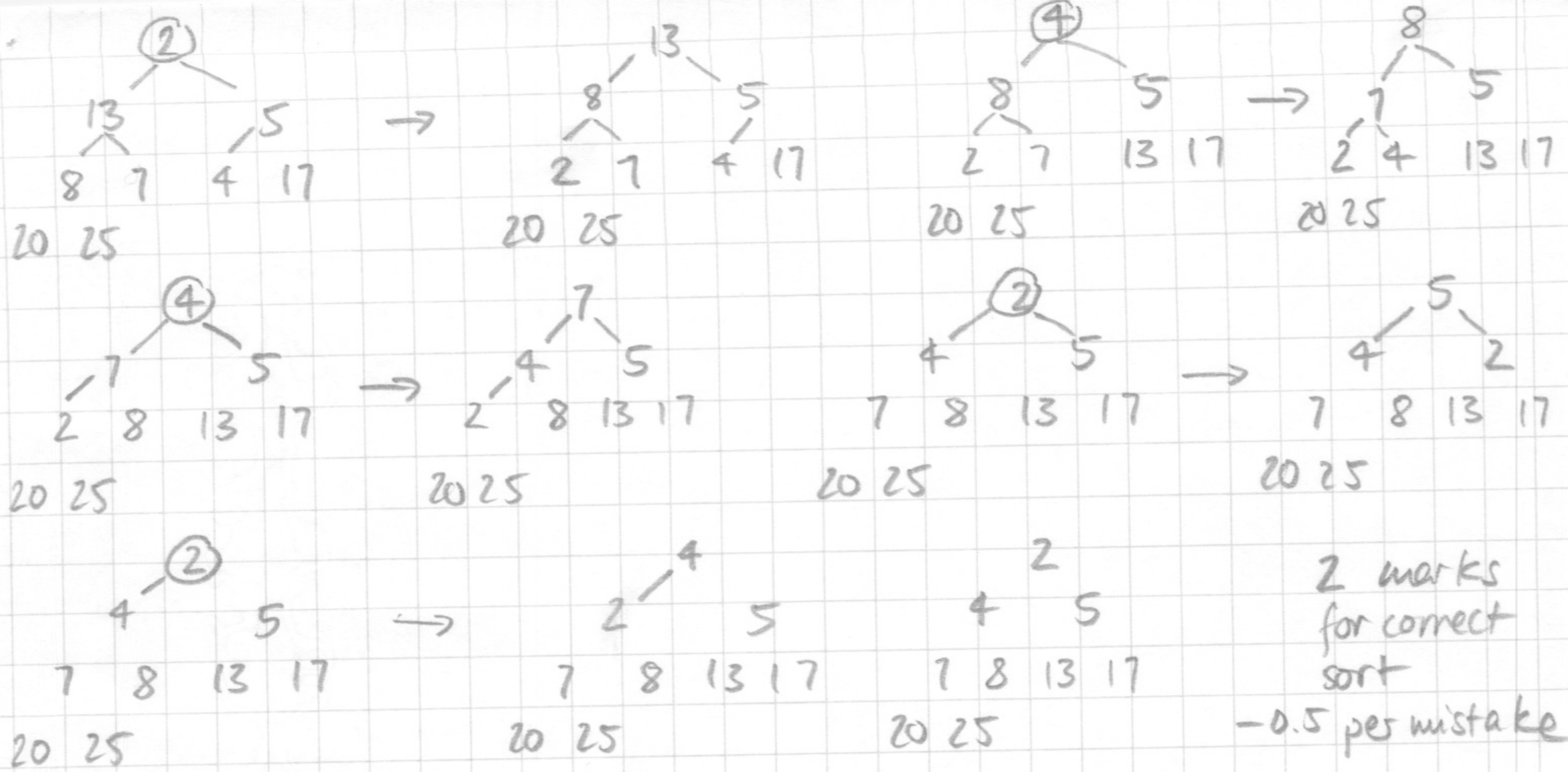


1 mark for correctly built maxheap

-0.5 per mistake

If students built wrong maxheap, they can still get marks for correct sorting:





2 marks for correct sort
-0.5 per mistake

Problem 2

(a) In every pass/round, the algorithm needs to make exactly one comparison. There are $N-1$ passes/rounds. The running time hence is $\Theta(N)$.
[1 mark for $\Theta(N)$, 1 mark for explanation]

(b) At best, we can assume the algorithm for merging always chooses the left of two equal elements first.
We then get $\Theta(\log N)$ many stages of merging.
In each stage, we need to make $\frac{N}{2} = \Theta(N)$ many comparisons.
$\Rightarrow$ The running time is $\Theta(N \log N)$.

[1 mark for $\Theta(N \log N)$, 2 marks for explanation]

Problem 3 [1 mark for each answer]

stable: Insertion Sort, Mergesort
not stable: Shellsort, Heapsort

Problem 4 [each 1 mark for correct answer (Θ-bound), 1 mark for explanation]

(a) $T(N) = 2T(\frac{N}{4}) + 1$

$a=2, b=4, f(N) = 1 = N^0, \quad z = \log_b a = \frac{1}{2} > 0$
Master Theorem (1) $\Rightarrow T(N) = \Theta(N^z) = \Theta(N^{\frac{1}{2}}) = \Theta(\sqrt{N})$

(b) $T(N) = 2T(\frac{N}{4}) + \sqrt{N}$

$a=2, b=4, f(N) = \sqrt{N} = N^{\frac{1}{2}}, z = \log_b a = \frac{1}{2}$
MT (2) $\Rightarrow T(N) = \Theta(N^z \log N) = \Theta(N^{\frac{1}{2}} \log N) = \Theta(\sqrt{N} \log N)$

(c) $T(N) = 2T(\frac{N}{4}) + N^2$

$a=2, b=4, f(N) = N^2, \quad z = \log_b a = \frac{1}{2} < 2$
Moreover, there is $n_0 \in \mathbb{N}$, $c<1$ such that $2 \cdot \frac{N^2}{4} \leq c \cdot N^2$ for all $N \geq n_0$, namely $c = \frac{1}{2}$, $n_0 = 1$.
MT (3) $\Rightarrow T(N) = \Theta(f(N)) = \Theta(N^2)$.

(d) $T(N) = 9T(\frac{N}{3}) + N$

$a=9, b=3, f(N) = N^1, \quad z = \log_b a = 2 > 1$
MT (1) $\Rightarrow T(N) = \Theta(N^z) = \Theta(N^2)$.

Problem 5 $T(N) = 2T(\frac{N}{2}) + N \log N$ [2 marks]

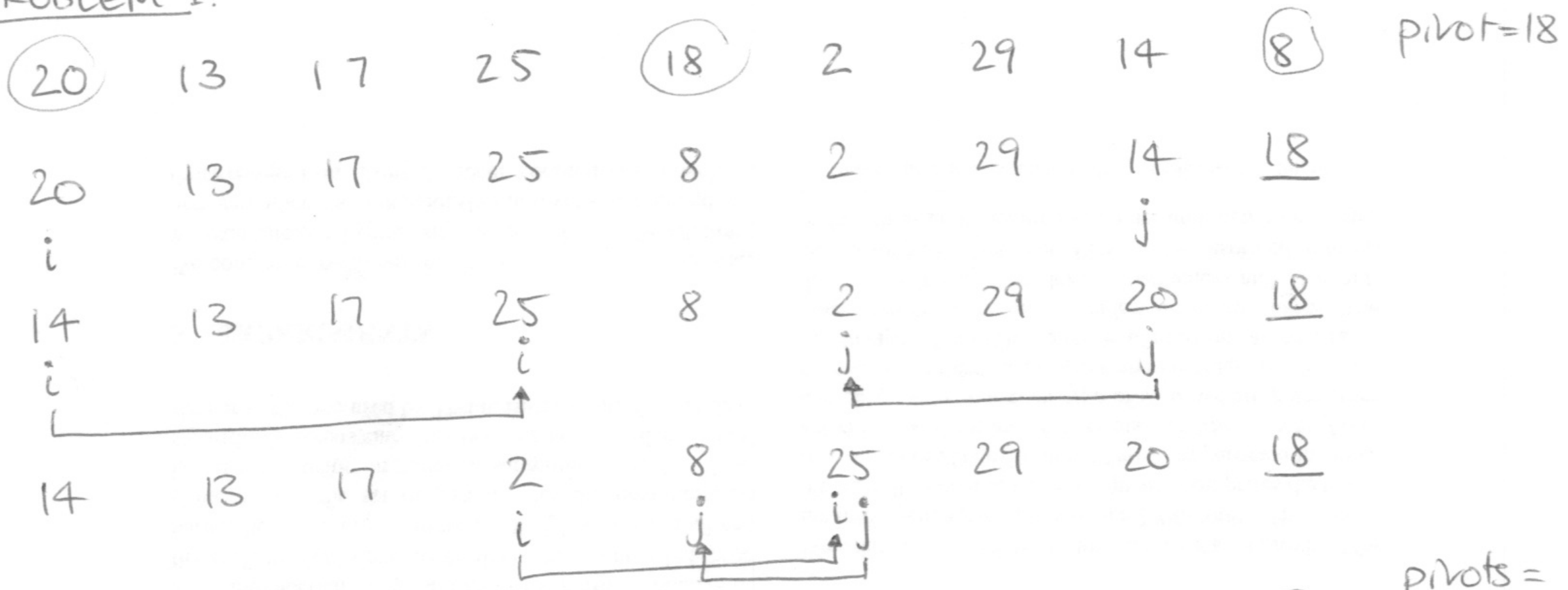
The problem is not that $f(N)$ is not equal to N^x for some x. $f(N)$ does not have to be equal to N^x for some x for the M.T. to apply!

$a=2, b=2, z=1. \quad f(N) = N \log N = \Omega(N^1) = \Omega(N^z)$

$f(N)$ is $\Omega(N^z)$, but there is no $x > z$ such that $f(N) = \Omega(N^z)$.
Therefore case (3) of M.T. does not apply.
Cases (1) and (2) don't apply, because $f(N) \neq O(N^z)$.

ASSIGNMENT 6

PROBLEM 1.



[subtract 1 mark for each mistake]

PROBLEM 2. The students have to find one correct topological ordering. There are several possibilities:

s, G, D, A, B, H, E, I, F, C, t
s, G, H, D, A, B, E, I, F, C, t
s, G, D, H, A, B, E, I, F, C, t
s, G, D, A, H, B, E, I, F, C, t
s, G, H, D, A, E, B, I, F, C, t
s, G, D, H, A, E, B, I, F, C, t
s, G, D, A, H, E, B, I, F, C, t
s, G, H, D, A, E, I, B, F, C, t
s, G, D, H, A, E, I, B, F, C, t
s, G, D, A, H, E, I, B, F, C, t
s, G, H, D, A, E, I, F, B, C, t
s, G, D, H, A, E, I, F, B, C, t
s, G, D, A, H, E, I, F, B, C, t

[subtract 1 mark for each mistake]

NOTE: the graph corresponding to the given adjacency list is the one shown in Figure 9.81 (page 437) of the textbook. This is a weighted graph, and the weights are given in the adjacency matrix, even though they are completely irrelevant to the problem. The students have to figure out themselves that the only thing that matters is where the edges are (and in which direction they go), but not what the weights of the edges are.

Some students might try to illustrate the algorithm with which they obtain the topological order, but it's not required for this question.

PROBLEM 3.

3.1 Suppose G has a cycle, $[w_1, \dots, w_n]$, where $w_1 = w_n$.
Assume, by way of contradiction, that G has a topological ordering $v_1, \dots, v_z$. Let $i, j \in \{1, \dots, z\}$ such that $w_1 = v_i$, $w_2 = v_j$.
Since $[w_1, w_2]$ is a path in G, and $v_1, \dots, v_z$ is a top. ordering, we have

$$i < j. \quad (*)$$

Since $[w_2, w_3, \dots, w_{n-1}, w_n]$ $(= [w_2, w_3, \dots, w_{n-1}, w_1])$ is a path in G, and $v_1, \dots, v_z$ is a top. ordering, we have $j < i$. $(**)$
$(*)$ and $(**)$ contradict each other. Therefore G has no top. ordering.

[give 1 mark if only a small part of the reasoning is missing]

3.2 If G has no top. ordering then there is a <u>subgraph</u> of G in which <u>no</u> vertex has in-degree zero (otherwise the alg. given in class produces a top. ordering for G). Let $v_1, \dots, v_k$ be the vertices in this subgraph.
For each $v \in \{v_1, \dots, v_k\}$ there is a $\text{pred}(v) \in \{v_1, \dots, v_k\}$ such that

$$(\text{pred}(v), v) \in E.$$

Hence $[\underbrace{\text{pred}(\text{pred}(\dots \text{pred}(v)) \dots))}_{k}, \underbrace{\text{pred}(\dots \text{pred}(v) \dots)}_{k-1}, \dots, \text{pred}(v), v]$
is a path in G using only vertices in $\{v_1, \dots, v_k\}$. Since the path has length k, at least one vertex in $\{v_1, \dots, v_k\}$ occurs twice on this path.
Therefore the path contains a cycle as a subpath.
Hence G has a cycle.

[give 2 marks if only a small part of the reasoning is missing, give 1 mark — " — is there]

CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Assignment 7 – November 16, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

answer key

Problem 1. (4+4 marks) [The students need to give the correct table only for either (a) or (b). For the other part, it is enough if they give the solution. Subtract 1 mark for each mistake.]

(a)

	initially			after step 1			after step 2		
vertex	known	distance	previous	known	distance	previous	known	distance	previous
A		0	–	true	0	–	true	0	–
B		∞	–		5	A		5	A
C		∞	–		3	A	true	3	A
D		∞	–		∞	–		10	C
E		∞	–		∞	–		10	C
F		∞	–		∞	–		∞	–
G		∞	–		∞	–		∞	–

	after step 3			after step 4			after step 5		
vertex	known	distance	previous	known	distance	previous	known	distance	previous
A	true	0	–	true	0	–	true	0	–
B	true	5	A	true	5	A	true	5	A
C	true	3	A	true	3	A	true	3	A
D		10	C		10	C		9	E
E		8	B		7	G	true	7	G
F		∞	–		∞	–		8	E
G		6	B	true	6	B	true	6	B

	after step 6			after step 7		
vertex	known	distance	previous	known	distance	previous
A	true	0	–	true	0	–
B	true	5	A	true	5	A
C	true	3	A	true	3	A
D		9	E	true	9	E
E	true	7	G	true	7	G
F	true	8	E	true	8	E
G	true	6	B	true	6	B

results: shortest path to ...
A is [A], length 0
B is [A,B], length 5
C is [A,C], length 3
D is [A,B,E,D], length 9
E is [A,B,G,E], length 7
F is [A,B,G,E,F], length 8
G is [A,B,G], length 6.

(b)

vertex	initially dequeued	initially distance	initially previous	after step 1 dequeued	after step 1 distance	after step 1 previous	after step 2 dequeued	after step 2 distance	after step 2 previous
A		∞	–		∞	–		∞	–
B		0	–	yes	0	–	yes	0	–
C		∞	–		1	B	yes	1	B
D		∞	–		∞	–		2	C
E		∞	–		1	B		1	B
F		∞	–		∞	–		∞	–
G		∞	–		1	B		1	B
queue:	B			C,E,G			E,G,D		

vertex	after step 3 dequeued	after step 3 distance	after step 3 previous	after step 4 dequeued	after step 4 distance	after step 4 previous	after step 5 dequeued	after step 5 distance	after step 5 previous
A		∞	–		∞	–		3	D
B	yes	0	–	yes	0	–	yes	0	–
C	yes	1	B	yes	1	B	yes	1	B
D		2	C		2	C	yes	2	C
E	yes	1	B	yes	1	B	yes	1	B
F		2	E		2	E		2	E
G		2	B	yes	2	B	yes	2	B
queue:	G,D,F			D,F			F,A		

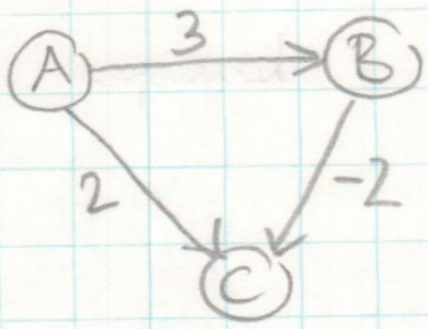
vertex	after step 6 dequeued	after step 6 distance	after step 6 previous	after step 7 dequeued	after step 7 distance	after step 7 previous
A		3	D	yes	3	D
B	yes	0	–	yes	0	–
C	yes	1	B	yes	1	B
D	yes	2	C	yes	2	C
E	yes	1	B	yes	1	B
F	yes	2	E	yes	2	E
G	yes	2	B	yes	2	B
queue:	A			empty		

results: shortest path to ...
A is [B,C,D,A], length 3
B is [B], length 0
C is [B,C], length 1
D is [B,C,D], length 2
E is [B,E], length 1
F is [B,E,F], length 2
G is [B,G], length 1.

CS340
2011/10

ASSIGNMENT 7

(2).



this graph has no cycle.

Dijkstra's alg. for source A will first set A as known with distance 0.
Then C will be marked with distance 2 and previous vertex A; B marked with distance 3 and previous vertex A. Since Dijkstra picks greedily, next it will set C known with distance 2.
C will not be updated again, since it is known. But there is a shorter path to C than [A, C], namely [A, B, C], which has length 1.

[2 marks for example, 2 marks for explanation]

(3) Since a shortest path contains at most $|V|-2$ many edges, each with weight in $\{1, 2, \dots, k\}$, the set of total possible lengths of shortest paths is $\{0, 1, 2, \dots, k\cdot(|V|-2)\} \cup \{\infty\}$.
Use an array of buckets numbered $0, 1, 2, \dots, k\cdot(|V|-2), k\cdot(|V|-2)+1$, used for storing vertices whose distance equals the bucket number ($k\cdot(|V|-2)+1$ means ∞).
→ [2 marks]

(Bucket(v) = k·(|V|−2) +1)

algorithm: (i) initially, place all vertices in bucket $k\cdot(|V|-2)+1$, except for vertex s in bucket 0. (Bucket(s) = 0) } O(|V|)

(ii) currentBucket = 0;

(iii) while currentBucket empty and currentBucket < $k\cdot(|V|-2)+1$ { currentBucket++; } } O(k·|V|) in total over time

(iv) for each v in currentBucket
{ for each $w \in V$ with $(v,w) \in E$ and Bucket(w) > currentBucket + c(v,w)
{
Bucket(w) = currentBucket + c(v,w);
w.previous = v;
}
}
currentBucket++;
goto (iii);

} O(|E|) in total over time

work on adjacency list

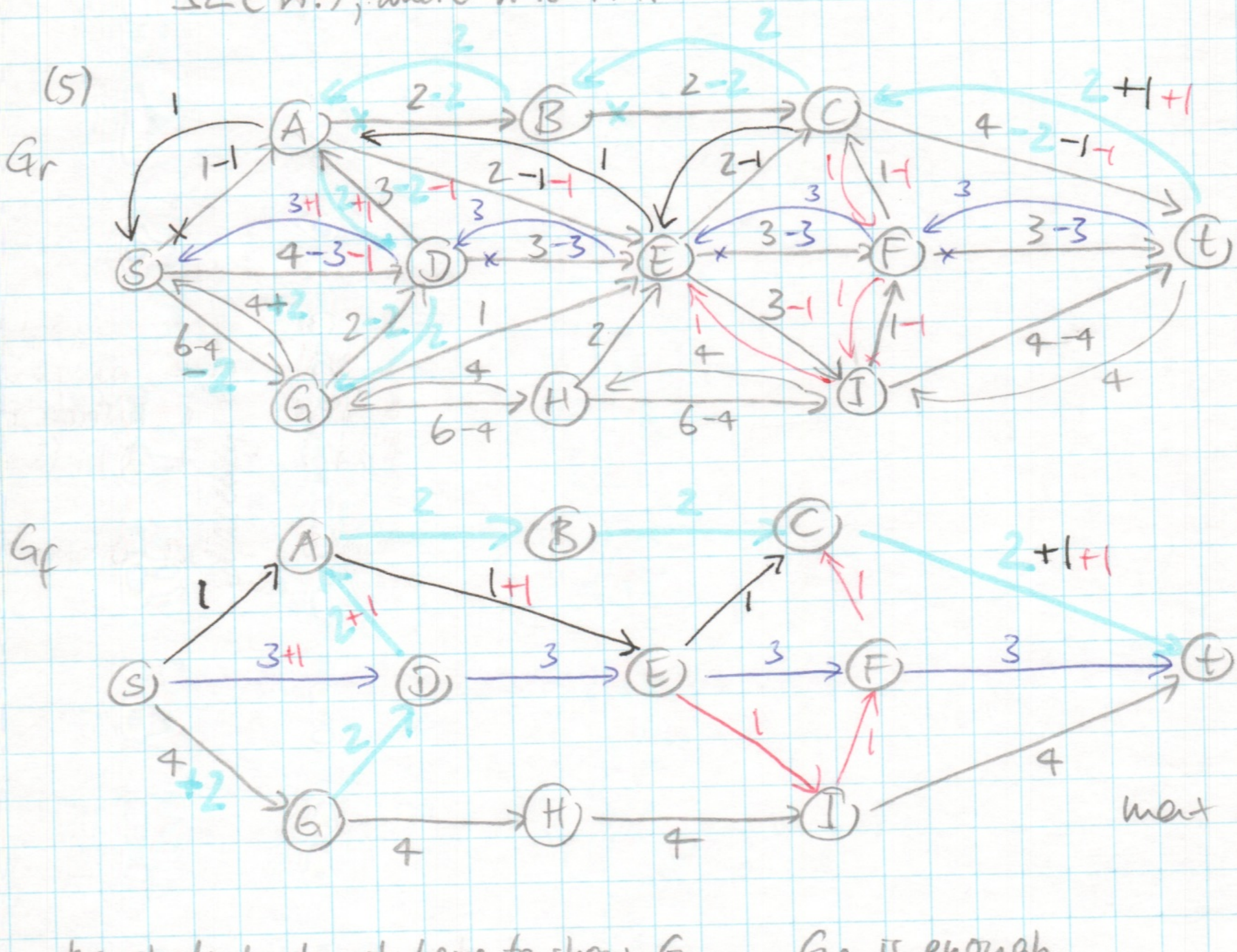
[3 marks for algorithm]
[1 mark for running time explanation]

(4) a. Since G is acyclic, G is a collection of trees.
Hence there is always at most one path from any vertex to any other vertex. [2 marks]

⇝ algorithm: BFS will work and has running time $O(|V|+|E|)$ [2 marks]

(b) In this case one would potentially have to check all possible paths from s to a vertex v. There could be exponentially many. [2 marks]
An algorithm that checks all paths would have cost of $\Omega(n!)$, where n is $|V|$. [1 mark]

(5)



the students do not have to show G_r; G_f is enough.

[give 3 marks if they find flow of 10
give 2 marks if they find flow of 7-9
give 1 mark if they find less flow, but try the right thing]

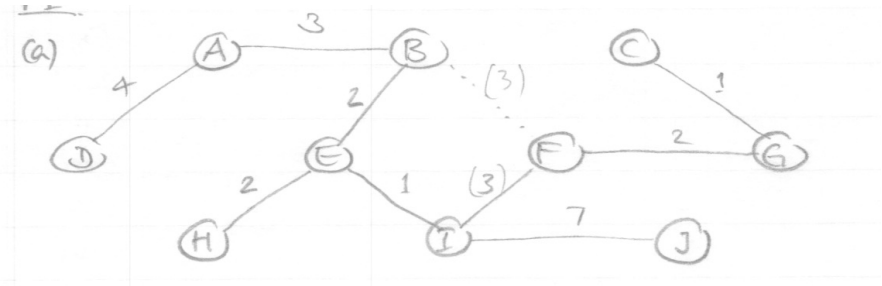
CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Assignment 8 – November 27, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

answer key

Problem 1 (6+1 marks).

(a) There are two options, as indicated in the following graphical representation of an MST:



[2 marks, even if only one of the two options is given.]

Order in which the edges are output by Prim's algorithm: [2 marks]

(A, B), (B, E), (E, I), (E, H), either (I, F) or (B, F), (F, G), (G, C), (A, D), (I, J)

Note: here I started at vertex A, but one could also start at any other vertex, which would then of course yield a different order.

Order in which the edges are output by Kruskal's algorithm: [2 marks]

(E, I), (G, C), (F, G), (E, H), (B, E), (A, B), either (I, F) or (B, F), (A, D), (I, J)

(b) No. Both algorithms have the choice between (I, F) and (B, F) at some point. Only one of the two edges will be in the MST. [1 mark]

Problem 2 (2+2 marks).

(a) A, B, C, E, G, D, F [2 marks]

(b) A, B, C, D, F, E, G [2 marks]

Problem 3 (4 marks). Such a vertex w corresponds to a row that contains only zeroes and a column that contains only ones (except for the entry on the diagonal, which is a zero).

Rough idea: Starting with $i = j = 0$, we parse the ith row from column j to the right as long as each entry is zero. If the jth entry in the ith row is a 1, then the vertex i is no longer a candidate for w. But for $j > i$, we know more than that: the vertices corresponding to indices k with $i < k \leq j - 1$ are no longer candidates either, because each of them have a zero in column k (off the diagonal). Then we go down in the matrix from position $[i][j]$ for as long as we see ones. If we hit a zero, we go right again, and so on. When we hit the right or bottom end of the matrix, the current row index or column index, respectively, is the only possible candidate for w, but still has to be verified.

In pseudocode:

1. Set $i = 0$ and $j = 0$.
2. While $i < |V|$ and $j < |V|$:
   if $A[i][j] = 0$, then $j++$, else $i++$.
3. If $j < V$, then check whether row j has all zeroes. If yes, return 'yes'.
4. If $i < V$, then check whether column i has all ones except in the diagonal. If yes, return 'yes'.

5. Return ‘no’.

Together, i and j are incremented at most $|V|$ times. The check after the while loop obviously takes time linear in $|V|$. Hence, the overall running time is $\Theta(|V|)$.

Problem 4 (3 marks).

The following reduction mapping will be used: Given a problem instance for $\mathcal{P}$, consisting of a graph $G = (V, E)$ and an integer $k \leq |V|$, the output of the reduction mapping is a problem instance for $\mathcal{P}'$, consisting of a graph $G' = (V', E')$ and an integer $k' \leq |V|$, where

- $k' = k$,
- $V' = V$,
- $E' = \{(v, w) \in V \times V \mid (v, w) \notin E\}$, i.e., E' makes exactly those pairs of vertices in V adjacent in G' that are *not* adjacent in the original graph G.

Since E' can be computed in time quadratic in $|V| + |E|$, this reduction mapping can be computed in polynomial time.

[1 mark for the correct mapping, 1 mark for explaining why it runs in polynomial time.]

It is now easy to see the following:

- If (G, k) is a positive instance of $\mathcal{P}$, then (G', k') is a positive instance of $\mathcal{P}'$. [0.5 marks]
- If (G, k) is a negative instance of $\mathcal{P}$, then (G', k') is a negative instance of $\mathcal{P}'$. [0.5 marks]