CS180-Spring20 Homework1

Gurbir Singh Arora

TOTAL POINTS

100 / 100

QUESTION 1

1 T or F 10 / 10

✓ **- 0 pts** **Correct**

- 2 pts Reasoning not clear or conclusion not clear

- 5 pts Reasoning is correct and answer is wrong

- 10 pts Incorrect answer and reasoning

- 8 pts Incorrect reasoning

QUESTION 2

2 Stable Matching 20 / 20

✓ **- 0 pts** **Correct**

- 2 pts 2.a Reasoning not clear or not complete

- 2 pts 2.b reasoning not clear or not complete

- 5 pts 2.a answer incorrect or not stated

- 10 pts 2.b Proof is incorrect

- 10 pts 2.a incorrect

QUESTION 3

3 Sort the functions 15 / 15

✓ **- 0 pts** **all correct**

- 1 pts 2<6

- 1 pts 2<1

- 1 pts 2<4

- 1 pts 2<5

- 1 pts 3<6

- 1 pts 3<1

- 1 pts 3<4

- 1 pts 3<5

- 1 pts 6<1

- 1 pts 6<4

- 1 pts 6<5

- 1 pts 1<4

- 1 pts 1<5

- 1 pts 4<5

- 1 pts 2<3

QUESTION 4

4 Complexity equivalence 18 / 18

✓ **+ 6 pts** **Yes. From (1) we have g(n) ⍰= 1/c * f(n) for n >= n0, so g(n) = omega(f(n)).**

✓ **+ 6 pts** **Yes. From (1) we have f (n) ⍰* g(n) <= c * g(n)^2 for n >= n0, so f(n)g(n) = omega(g(n)^2).**

✓ **+ 6 pts** **No. For example,...**

+ 0 pts Nothing written

QUESTION 5

5 Palindrome 12 / 12

✓ **- 0 pts** **Correct**

- 2 pts Inaccurate expression

- 5 pts Memory used not O(1)

- 2 pts need to consider the cases when n is odd or even

- 2 pts We don't assume we know the length of list

- 12 pts Wrong Algorithm

- 6 pts Not reversing half of the list

QUESTION 6

6 Heap 25 / 25

✓ **+ 5 pts** **1 6 2 8 12 14 5 20 10 15**

✓ **+ 5 pts** **20 15 12 10 14 2 1 5 6 8**

✓ **+ 15 pts** **Correct Algorithm**

+ 0 pts 0

Gurbir Arora
105178554

CS 180 HW #1

1) True and proved by contradiction

Suppose as stable matching S, w/ m ranked ~~1st~~ 1st on the preference list of w & w ranked 1st on the preference list of m, w/ pairs (m, w') & (m', w). Since w is ranked 1st on m pref list, m prefers w to w', and since m is ranked first on w's list, w prefers m to m'. This is an instability thus proving that this is a contradiction, meaning that the pair (m, w) must belong to S.

1 T or F 10 / 10

✓ - **0 pts** **Correct**

- **2 pts** Reasoning not clear or conclusion not clear
- **5 pts** Reasoning is correct and answer is wrong
- **10 pts** Incorrect answer and reasoning
- **8 pts** Incorrect reasoning

2)

m_1, m_2 & w_1, w_2. w/ n men & n women

m_1's pref: $w_1 > w_2 > \ldots$
m_2's pref: $w_2 > w_1 > \ldots$
w_1's pref: $m_2 > m_1 > \ldots$
w_2's pref: $m_1 > m_2 > \ldots$

a) Every execution of the G-S algorithm results in m_1 matching w/ w_1 & m_2 w/ w_2

Proof: Suppose by way of contradiction, some execution of the G-S algorithm results in a matching where m_1 matches w/ w_2 & m_2 w/ w_1. Now, since men propose in decreasing order of preference this means at some point a man is rejected by a valid partner during this execution. Consider the moment when this man, m_1, is rejected by valid partner w_2. Since men propose in decreasing order of preference, & since this is the first valid partner rejection, it must be the case that w_2 is m_1's best valid partner.

This rejection may have happened because: (1) m_1 proposed and was turned down in favor of w_2's existing engagement. (2) because w_2 broke her engagement to m_1 in favor of a better proposal. In either case, w_2 forms or continues an engagement w/ a man, m_2, who she prefers to m_1. Since w_2 is a valid partner of m_1, there exists a stable matching containing (m_1, w_2). Now, m_2 is paired w/ w_2.

Since the rejection of m_1 by w_2 was the first rejection of a man by valid partner in this execution, it must be the case that

m2 hasn't been rejected by a valid partner since the point he was engaged to w. Since he proposed in decreasing order of preference and since w_2 is clearly a valid partner of m_2, it must be that m_2 prefers w_1 to w_2. We know that w_1 prefers m_2 to m_1 B since the pair (m_2, w_1) are not in S', it must follow that (m_2, w_1) is an instability in S'.

This is a contradiction and proves that every execution results in m_1 matching to w_1 B m_2 matching w/ w_2.

- From the man proposing GS algorithm, we know that since m1 has a pref of w1 B m2 of w2, m1 will propose to w1 first on since she is free, she will accept. Similarly, m2 B w2 will engage.

b) Show that every stable matching m_1, m_2 are matched to w_1, w_2 ie $(m_1, w_1)(m_2, w_2)$ or (m_1, w_2), (m_2, w_1)

In part A, we proved the first half of this by showing that running G-S w/ men proposing will always result in (m_1, w_1) B (m_2, w_2). Now, if we run the G-S w/ women proposing it is quite intuitive that every execution will result in (m_1, w_2), (m_2, w_1) by following the steps in part A and switching w_1 for m_1 B w_2 for m_2 and v.v.

This proves that every stable matching will result in either (m_1, w_1), (m_2, w_2) or (m_1, w_2), (m_2, w_1)

Let us suppose, by way of contradiction, that some execution of the G-S algorithm results in a matching S in which some woman is paired with a man who is not her best valid partner. Since women propose in decreasing order of preference, this means that some woman is rejected by a valid partner during the execution of the algorithm. So consider the first moment during the execution g in which some woman, say w1, is rejected by a valid partner m1. Again, since women propose in decreasing order of preference, and since this is the first time such rejection has occurred, it must be that m1 is w1's best valid partner. The rejection of w1 by m1 may have happened because:

1. w1 proposed and was turned down in favor of m1's existing engagement
2. m1 broke her engagement to w1 in favor of a better proposal.

But either way, at this moment m1 forms or continues an engagement with a woman w2 whom he prefers to w1. Since m1 is a valid partner of w1, there exists a stable matching S' containing the pair (m1, w1). Suppose w2 is paired with a man m2. Since the rejection of w1 by m1 was the first rejection of a woman by a valid partner in the execution, it must be that w2 had not been rejected by any valid partner at the point in when she became engaged to m1. Since she proposed in decreasing order of preference, and since m2 is clearly a valid partner of w2, it must be that w2 prefers m1 to m2. But we have already seen that m1 prefers w2 to w1, for in execution he rejected w1 in favor of w2. Since (w2, m1) S', it follows that (w2, m1) is instability in S'. This contradicts our claim that S' is stable and hence contradicts our initial assumption.

Since we have proved that the outcome of any GS execution(man or woman proposal) will result in, (m1 w1),(m2, w2) or (m1, w2),(m2, w1) and our GS executions would cover every potential stable matching, we have successfully proved that this is the case for every stable matching.

2 Stable Matching 20 / 20

✓ **- 0 pts** **Correct**

- 2 pts 2.a Reasoning not clear or not complete

- 2 pts 2.b reasoning not clear or not complete

- 5 pts 2.a answer incorrect or not stated

- 10 pts 2.b Proof is incorrect

- 10 pts 2.a incorrect

3)

$f_1(n) = n^{2.5}$ $\quad f_4(n) = 10^n$

$f_2(n) = \sqrt{2n}$ $\quad f_5(n) = 100^n$

$f_3(n) = n + 10$ $\quad f_6(n) = n^2 \log n$

we know that the exponentials $f_4(n)$ & $f_5(n)$ grow the fastest & that $f_5(n)$ grows faster than $f_4(n)$ bc it can be written as $f_5(n) = 100^n = 10^{2n}$ & at a higher level $10 < 100$.

so now the list looks like:

$< f_4(n) < f_5(n)$

now, $f_1(n) = n^{2.5} = n^2 \cdot \sqrt{2n}$ & $f_2(n) = \sqrt{2n} = (2n)^{0.5}$

It is obvious that $f_2(n)$ is the slowest bc it has the smallest degree. So, our list looks like

$f_2(n) < \ldots < f_4(n) < f_5(n)$

Now, in order to order $f_1(n)$, $f_3(n)$ & $f_6(n)$

$\lim_{n\to\infty} n^{2.5} = \infty^{2.5} \Rightarrow \lim_{n\to\infty} 2.5\, n = 2.5(\infty) = \infty$

$\lim_{n\to\infty} n + 10 = \infty + 10 = \infty$

$\lim_{n\to\infty} n^2 \log n = \lim_{n\to\infty} (2n \log n + n)$

$f_3(n)$ has a degree of 1, so logically

$f_2(n) < f_3(n) \ldots < f_4(n) < f_5(n)$

we can tell that $f_6(n) < f_1(n)$ because $f_6 = n^2 \log(n)$ & $f_1 = n^2 \sqrt{2n}$ & $\log(n) = O(\sqrt{2n})$

Therefore, we get the order

$f_2(n) < f_3(n) < f_6(n) < f_1(n) < f_4(n) < f_5(n)$

3 Sort the functions **15 / 15**

✓ **- 0 pts** **all correct**

- **- 1 pts** 2<6
- **- 1 pts** 2<1
- **- 1 pts** 2<4
- **- 1 pts** 2<5
- **- 1 pts** 3<6
- **- 1 pts** 3<1
- **- 1 pts** 3<4
- **- 1 pts** 3<5
- **- 1 pts** 6<1
- **- 1 pts** 6<4
- **- 1 pts** 6<5
- **- 1 pts** 1<4
- **- 1 pts** 1<5
- **- 1 pts** 4<5
- **- 1 pts** 2<3

4) a. $g(n) = \Omega(f(n))$

True

proof: as $f(n)$ is $O(g(n))$ there is a constant $c>0$ & $n \in \mathbb{N}$ s.t. $f(n) \leq c\, g(n)$ $\forall$ $n \geq \mathbb{N}$

rearranging this inequality, we get

$$\frac{1}{c} \cdot f(n) \leq g(n) \quad \forall \; n \geq \mathbb{N}$$

now, let $k = \frac{1}{c}$, so we can rewrite

$$g(n) \geq k \cdot f(n) \quad \forall \; n \geq \mathbb{N}$$

As $c>0$, $k>0$ this inequality shows that $g(n) \geq \Omega(f(n))$

b. $f(n) \cdot g(n) = O(g(n)^2)$ True

proof: there $\exists$ some n_0 $\forall$ $n > n_0$ s.t. $f(n) < \epsilon \cdot g(n)$, where ϵ is some constant. We can say that for any positive n, $f(n) \cdot g(n) < c(g(n))^2$ for all $c>0$. Hence, $f(n) \cdot g(n) = O(g(n)^2)$

c. $2^{f(n)} = O(2^{g(n)})$ false

proof by counter example:

let $f(n) = 2n$ & $g(n) = n$. This holds under the assumption that $f(n)$ is $O(g(n))$ $\forall n$, where $2n < Cn$, where C is constant & $C>3$. However, 2^{2n} is not $O(2^n)$ in the case where $f(n) = 2n$ & $g(n) = n$ since $2^{2n} >> C * 2^n$, which we can simplify to $1 > C * 0$ for $n>0$ & any constant C. Thus, $2^{f(n)} \neq O(2^{g(n)})$

more precisely: let $f(n) = 2\log n$ & $g(n) = \log(n)$

$2\log n \leq C \log(n)$, so $f(n) = O(g(n))$

$2^{f(n)} = 2^{\log(n^2)} = n^2$

$2^{g(n)} = 2^{\log(n)} = n$ & $n^2 \neq O(n)$

4 Complexity equivalence 18 / 18

✓ + 6 pts Yes. From (1) we have g(n) ≥= 1/c * f(n) for n >= n0, so g(n) = omega(f(n)).

✓ + 6 pts Yes. From (1) we have f (n) * g(n) <= c * g(n)^2 for n >= n0, so f(n)g(n) = omega(g(n)^2).

✓ + 6 pts No. For example,...

+ 0 pts Nothing written

5) Given a list $a_1, \ldots, a_n$ check if palindrome in O(n) time & O(1) additional memory.

1) Get the middle of the linked list

2) Reverse second half of the linked list

3) Check if the first half & second half are identical

4) Construct the original linked list by reversing the second half and attaching it back to the first half

I will write the code/pseudo code of the main function of this problem

```
bool isPalindrome(struct Node* head)  {
    struct Node *slow_ptr = head, *fast_ptr = head;
    struct Node *second_half, *prev_of_slow_ptr = head;
    struct Node* midnode = NULL; // for odd size list
    bool res = true;
    if (head != NULL && head->next != NULL) {
        /* Get the middle of the list by moving slow_ptr by 1
          and fast_ptrr by 2 */
        while (fast_ptr != NULL && fast_ptr->next != NULL) {
            fast_ptr = fast_ptr->next->next;

            /* Previous of the slow_ptr needed for
             linked lists with odd elements */
            prev_of_slow_ptr = slow_ptr;
            slow_ptr = slow_ptr->next;
        }
        /* fast_ptr would become NULL when there are even elements in the list.
           And not NULL for odd elements. We need to skip the middle node
           for odd case and store it somewhere so that we can restore the
           original list*/
        if (fast_ptr != NULL) {
            midnode = slow_ptr;
            slow_ptr = slow_ptr->next;
        }
        // Now reverse the second half and compare it with first half
        second_half = slow_ptr;
        prev_of_slow_ptr->next = NULL; // NULL terminate first half
        reverse(&second_half); // Reverse function to reverse the second half
        res = compareLists(head, second_half); // compares to see if first half and second are
equal
         /* Construct the original list back */
        reverse(&second_half); // Reverse the second half again
        // If there was a mid node (odd size case) which
        // was not part of either first half or second half.
        if (midnode != NULL) {
            prev_of_slow_ptr->next = midnode;
            midnode->next = second_half;
        }
        else
            prev_of_slow_ptr->next = second_half;
    }
    return res;
}
```

5 Palindrome 12 / 12

✓ **- 0 pts** **Correct**

- 2 pts Inaccurate expression

- 5 pts Memory used not O(1)

- 2 pts need to consider the cases when n is odd or even

- 2 pts We don't assume we know the length of list

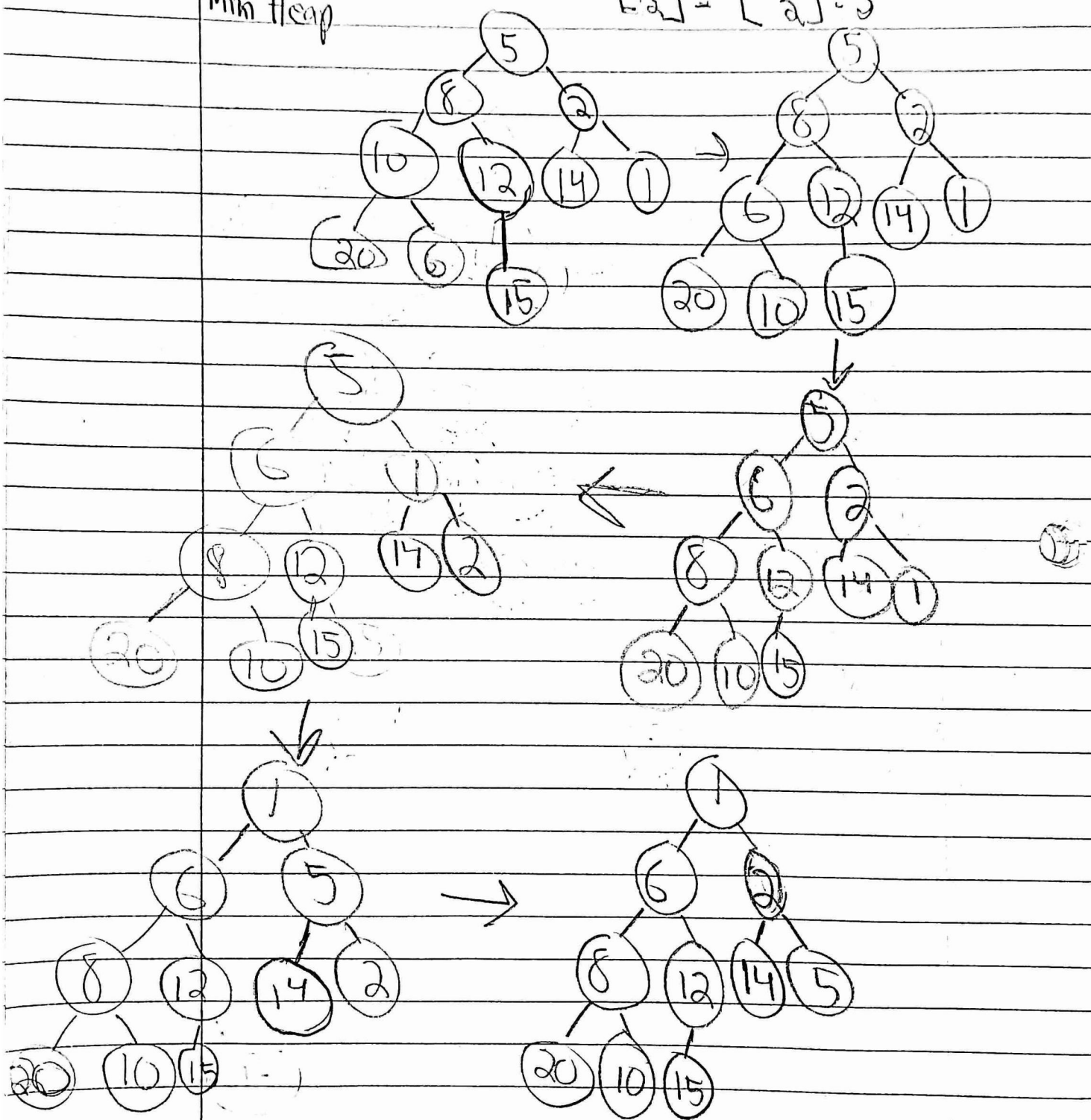
- 12 pts Wrong Algorithm

- 6 pts Not reversing half of the list

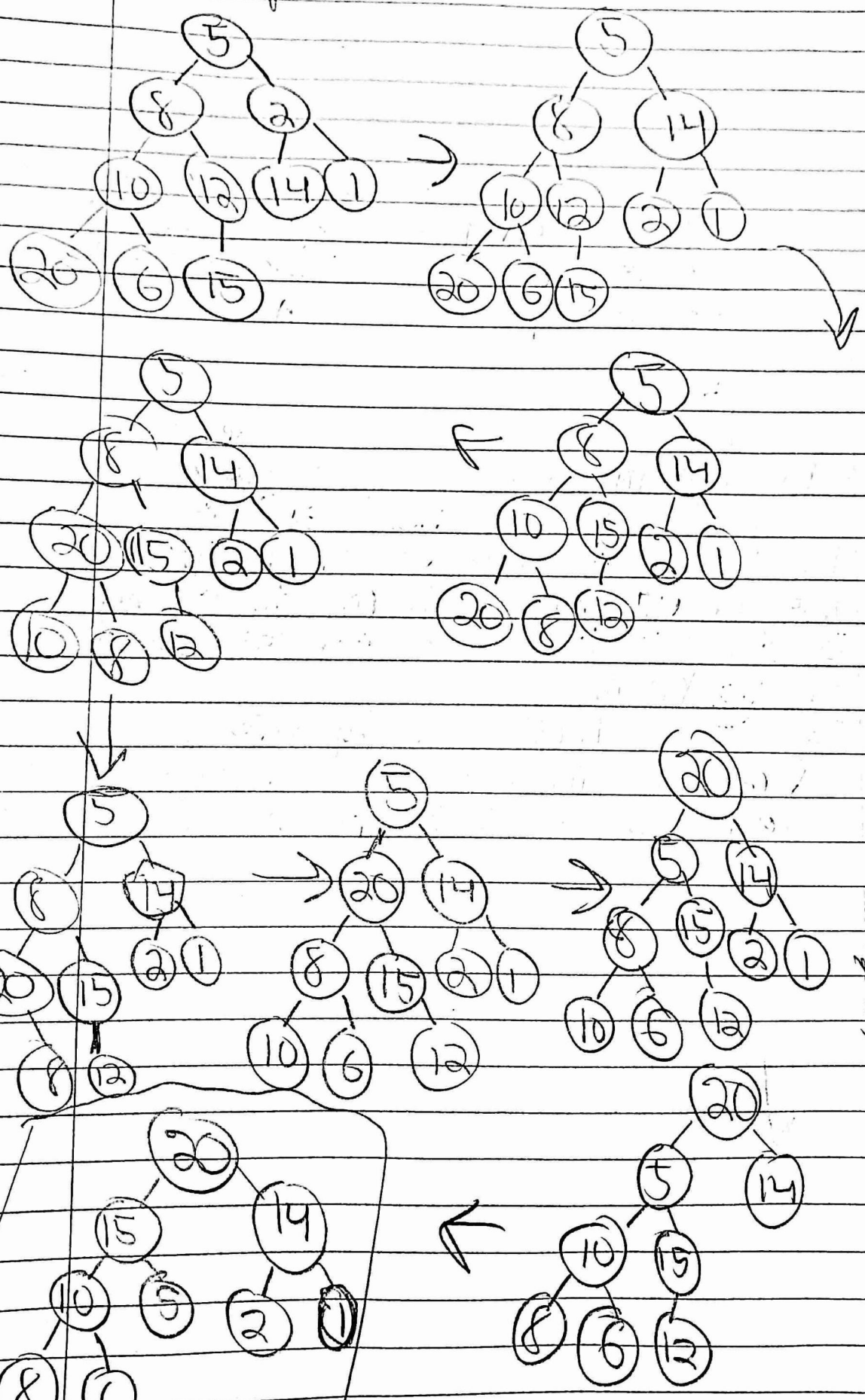
6) Given sequence [5, 8, 2, 10, 12, 14, 1, 20, 6, 15]

10 nodes ∴ $\lfloor \frac{N}{2} \rfloor = \lfloor \frac{10}{2} \rfloor = 5$

Min Heap



Max Heap



Medium - Heap

Basic Idea: 2 heaps: one min-heap & one max-heap with each heap containing $\frac{1}{2}$ of the data
Every element in min heap is greater than or equal to the median & every element in max-heap is less or equal to median

find-medium

If min heap contains one more element than the max heap, median is the top of the min-heap
If max heap contains one more element than min heap, median is in the top of the max heap
If both heaps contain the same # of elements, the total # of elements must be even, so you take the mean $(a_{n/2} + a_{n/2+1})/2$

push

To insert, compare new element w/ top of heaps and if new element is greater than the current median, it goes to the min heap.
If it is less than the current median, it goes to max heap.
If sizes of heaps differ by more than one element, extract the min/max from the heap with more elements and insert it into the other heap

6 Heap 25 / 25

- ✓ + 5 pts 1 6 2 8 12 14 5 20 10 15
- ✓ + 5 pts 20 15 12 10 14 2 1 5 6 8
- ✓ + 15 pts Correct Algorithm
  - + 0 pts 0