CPSC 331 HW4_Written

Ali Akbari

TOTAL POINTS

8.5 / 15

QUESTION 1

1 Applying Knuth's Formulas 1.5 / 2

+ **2 pts** Correct

✓ **+ 1.5 pts** **Correct formulas- some minor mistakes**

+ **1.5 pts** formulas for one of the strategies are missing or incorrect

+ **1 pts** formulas for two strategies are mostly missing or inorrect

- **0.5 pts** Very minor error

+ **0 pts** No submission (not done)

✓ **+ 0 pts** **inaccurate rounding (no point deducted)**

1 ?

2 ?

3 2.011

4 2.64

QUESTION 2

2 Correctness of Heapsort 2 / 5

+ **5 pts** Correct

+ **2 pts** Main Loop invariant: arr[i+1, .. n-1]ascending order (identification and proof)

- **0.5 pts** Main Loop Invariant: mostly correct: some error

✓ **+ 1 pts** **Main Loop Invariant : partially correct**

+ **0.5 pts** Main Loop Invariant: mostly incorrect - some correct parts

+ **0 pts** Main Loop invariant: Incorrect or missing

+ **1 pts** Helper Loop Invariant 1: arr[0, ..., i] a max-heap (identification and proof)

✓ **+ 0.5 pts** **Helper Loop Invariant 1: partially correct arr[0, ..., i] a max-heap (identification and proof)**

+ **0 pts** Helper Loop Invariant 1: Incorrect or missing

+ **1 pts** Helper Loop Invariant 2: arr\[0,..,i] <= arr\[i+1, ..., n-1] or arr\[0,..i] <=arr\[i+1](identification and proof)

✓ **+ 0.5 pts** **Helper Loop invariant 2: partially correct**

+ **0 pts** Helper Loop Invariant 2: Incorrect or missing

+ **1 pts** Partial Correctness: Correct

+ **0.5 pts** Partial Corretness: partially correct

✓ **+ 0 pts** **Partial Correctness: Incorrect or missing**

- **0.5 pts** Miscellaneous: minor mistake or missing

+ **0.5 pts** Reasonable Attempt

+ **0 pts** Not Submitted or Incorrect

5 ?

6 why true?

QUESTION 3

3 KQuickSort 2.5 / 5

+ **5 pts** Correct

✓ **+ 1.5 pts** **part a: recurrence is correct**

+ **1 pts** part a partially : a mistake or missing

+ **0.5 pts** part a: mostly incorrect, some correct parts

+ **0 pts** part a: recurrence completely incorrect or missing

+ **0.5 pts** part b: Usage of constants during substitution and in recurrence

✓ **+ 0 pts** **part b: incorrect usage or missing using some constants during substitutions and in recurrence**

+ **0.5 pts** part b: correct Substitution of T(k-1)

✓ **+ 0 pts** **part b: Incorrect or incomplete substitution of T(k-1)**

+ **1 pts** part b: Correct Substitution of T(n-k)

+ **0.5 pts** part b: Partially correct Substitution of T(n-k)

✓ **+ 0 pts** **part b: Incorrect or missing substitution of**

T(n-k)

+ **0.5 pts** part b: Correct recurrence solution

✓ **+ 0 pts part b: Incorrect or missing final recurrence Solution**

✓ **+ 0.5 pts part b: Correct asymptotic notation O(n^2) based on the recurrence solution**

+ **0 pts** part b: Incorrect or missing Asymptotic Notation or incorrect related arguments or not based on the recurrence solution

✓ **+ 0.5 pts part c: Correct**

- **0.5 pts** Miscellaneous: minor mistakes or missing

+ **0 pts** part b: not submitted or totally wrong

+ **0 pts** part c: completely incorrect or missing or based on incorrect arguments from previous section or not base on part b

+ **0 pts** Not submitted or totally wrong

7 cannot ignore constant at this point

8 3?

9 ?

QUESTION 4

4 Number of Tree Edges 2.5 / 3

+ **3 pts** Correct

✓ **+ 0.5 pts Base case correct**

+ **0 pts** Base case incorrect or missing

+ **0.5 pts** Correct Inductive hypothesis

✓ **+ 0 pts Incorrect or missing inductive hypothesis**

✓ **+ 2 pts Inductive step correct**

+ **1 pts** Inductive step partially correct

+ **0.5 pts** Inductive step : not convincing and mostly incorrect

+ **0 pts** Inductive Step totally incorrect or missing

- **0.5 pts** A minor mistake in Inductive step

+ **0 pts** Totally Incorrect or not submitted

+ **0 pts** Very minor mistakes (no points deducted)

10 we cannot conclude this from the I.H.

11 the same.

CPSC 331 Assingment #4 Written **Ali Akbari 30010402**

1)

	Linear Probing	Quadratic Probing	Double Hashing
Successful - $\frac{3}{4}$	$\frac{1}{2}(1+(\frac{1}{1-\frac{3}{4}}) = 2.5 \approx 3$	$1 - ln(1-\frac{3}{4}) - \frac{\frac{3}{4}}{2} = 0.8862 \approx 1$	$\frac{1}{\frac{3}{4}} ln(\frac{1}{1-\frac{3}{4}}) = 1.84839 \approx 2$
Unsuccessful - $\frac{3}{4}$	$\frac{1}{2}(1+(\frac{1}{(1-\frac{3}{4})^2}) = 8.5 \approx 9$	$\frac{1}{1-\frac{3}{4}} - \frac{3}{4} - ln(1-\frac{3}{4}) = 4.63629 \approx 5$	$\frac{1}{1-\frac{3}{4}} = 4 \approx$
Successful - $\frac{7}{8}$	$\frac{1}{2}(1+(\frac{1}{1-\frac{7}{8}}) = 4.5 \approx 5$	$1 - ln(1-\frac{7}{8}) - \frac{\frac{7}{8}}{2} = 1.3294 \approx 2$	$\frac{1}{\frac{7}{8}} ln(\frac{1}{1-\frac{7}{8}}) = 2.37650 \approx 3$
Unsuccessful - $\frac{7}{8}$	$\frac{1}{2}(1+(\frac{1}{(1-\frac{7}{8})^2}) = 32.5 \approx 33$	$\frac{1}{1-\frac{7}{8}} - \frac{7}{8} - ln(1-\frac{7}{8}) = 9.2044 \approx 10$	$\frac{1}{1-\frac{7}{8}} = 8 \approx$

*The approximate equal value is the rounded up number for the upper bound of probing.

This question deals with the heapsort algorithm shown below.

```
function Heapsort(arr)
    Heapify array arr;
    for i = n − 1 down to 1 do
        swap arr[i] with arr[0];
        restore heap property for the tree arr[0], ..., arr[i − 1] by percolating down the root;
    end
end
```

2)

Precondition:

An array that can be sorted, i.e has elements that can be compared by value.

Loop Invariants:

At a certain i^{th} iteration the following conditions should be met.

- Array keeps heap properties, i.e is a heap.
- $Array[n-1] \geq Array[n-2] \geq ..... \geq Array[i+2] \geq Array[i+1]$
- $for\ somer\ iteration\ of\ k = i,\ Array[k] \geq Array[j]\ where,\ i+1 \leq k \leq n-1\ \&\ 0 \leq j \leq i.$

Initialization:

i is initialized to n - 1. Heapify turns the array into a heap, the array after the heapify should be:

$Array[n-1] \geq ... \geq Array[i+1]\ which\ is\ true\ \ and$

$Array[k] \geq Array[j]\ where,\ n \leq k \leq n-1\ \&\ 0 \leq j \leq n-1$

So the loop invariant holds.

1 Applying Knuth's Formulas 1.5 / 2

+ **2 pts** Correct

✓ **+ 1.5 pts** **Correct formulas- some minor mistakes**

+ **1.5 pts** formulas for one of the strategies are missing or incorrect

+ **1 pts** formulas for two strategies are mostly missing or inorrect

- **0.5 pts** Very minor error

+ **0 pts** No submission (not done)

✓ **+ 0 pts** **inaccurate rounding (no point deducted)**

1. ?
2. ?
3. 2.011
4. 2.64

Maintenance:

Suppose the loop invariant holds for an iteration i. From initialization we know that the array is a heap from the initialization step, where heapify was called. So the root of the tree which is Array[0] = max element. Then after the swap the max element is changed to Array[i] = max element.

From our assumption:

- $Array[n-1] \geq ..... \geq Array[i+1] \geq Array[i]$

Once the percolate down method finishes we get
Array[0,1,...,i-1]

Therefore the loop invariant is held during the maintenance step.

Postcondition:

Given heap array is sorted so,

$Array[n-1] \geq Array[n-2] \geq ..... \geq Array[1] \geq Array[0]$

Corectness/Termination:

The loop runs from i = n - 1 and stops at i = 1. Using the $Array[k] \geq Array[j]$ condition we can see that the array is indeed sorted and in the maintenance step this postcondition is also proved.

Since postcondition holds and

3)

Let $T(n)$ denote the worst case number of steps to run KQuickSort on an array of size n.

Assume that KPartition takes $\Theta(n)$ steps to partition an array of size n.

a)

$T(n) = \Theta(n^2)$ *when* $n < k$ *we get the worst case since insertion sort is called.*

For $n \geq k$ three operations occur,

KPartition is called and the complexity of that is given as $\Theta(n)$.

Also two recursive calls are made to KQuickSort, one for the left partition and one for the right partition.

For the right partition the recursive call is done for n-k.

For the left partition the recursive call is done for k-1.

So the recurrence relation can be made:

$$T(n) = \begin{cases} \Theta(n^2) & n < k \\ \Theta(n) + T(n-k) + T(k-1) & n \geq k \end{cases}$$

2 Correctness of Heapsort 2 / 5

+ **5 pts** Correct
+ **2 pts** Main Loop invariant: arr[i+1, .. n-1]ascending order (identification and proof)
- **0.5 pts** Main Loop Invariant: mostly correct: some error

✓ + **1 pts** **Main Loop Invariant : partially correct**

+ **0.5 pts** Main Loop Invariant: mostly incorrect - some correct parts
+ **0 pts** Main Loop invariant: Incorrect or missing
+ **1 pts** Helper Loop Invariant 1: arr[0, ..., i] a max-heap (identification and proof)

✓ + **0.5 pts** **Helper Loop Invariant 1: partially correct arr[0, ..., i] a max-heap (identification and proof)**

+ **0 pts** Helper Loop Invariant 1: Incorrect or missing
+ **1 pts** Helper Loop Invariant 2: arr\[0,..,i] <= arr\[i+1, ..., n-1] or arr\[0,..i] <=arr\[i+1](identification and proof)

✓ + **0.5 pts** **Helper Loop invariant 2: partially correct**

+ **0 pts** Helper Loop Invariant 2: Incorrect or missing
+ **1 pts** Partial Correctness: Correct
+ **0.5 pts** Partial Corretness: partially correct

✓ + **0 pts** **Partial Correctness: Incorrect or missing**

- **0.5 pts** Miscellaneous: minor mistake or missing
+ **0.5 pts** Reasonable Attempt
+ **0 pts** Not Submitted or Incorrect

5 ?

6 why true?

Maintenance:

Suppose the loop invariant holds for an iteration i. From initialization we know that the array is a heap from the initialization step, where heapify was called. So the root of the tree which is Array[0] = max element. Then after the swap the max element is changed to Array[i] = max element.

From our assumption:

- $Array[n-1] \geq ..... \geq Array[i+1] \geq Array[i]$

Once the percolate down method finishes we get
Array[0,1,...,i-1]

Therefore the loop invariant is held during the maintenance step.

Postcondition:

Given heap array is sorted so,

$Array[n-1] \geq Array[n-2] \geq ..... \geq Array[1] \geq Array[0]$

Corectness/Termination:

The loop runs from i = n - 1 and stops at i = 1. Using the $Array[k] \geq Array[j]$ condition we can see that the array is indeed sorted and in the maintenance step this postcondition is also proved.

Since postcondition holds and

3)

Let $T(n)$ denote the worst case number of steps to run KQuickSort on an array of size n.

Assume that KPartition takes $\Theta(n)$ steps to partition an array of size n.

a)

$T(n) = \Theta(n^2)$ *when* $n < k$ *we get the worst case since insertion sort is called.*

For $n \geq k$ three operations occur,

KPartition is called and the complexity of that is given as $\Theta(n)$.

Also two recursive calls are made to KQuickSort, one for the left partition and one for the right partition.

For the right partition the recursive call is done for n-k.

For the left partition the recursive call is done for k-1.

So the recurrence relation can be made:

$$T(n) = \begin{cases} \Theta(n^2) & n < k \\ \Theta(n) + T(n-k) + T(k-1) & n \geq k \end{cases}$$

b)

Use iterative substitution to find an upper bound of $T(n)$.

Assume n is a multiple of K, so $n = m * K$

First we must work out a base case:

For $T(0) = ?$

Since n = 0 and that also means k = 0.

So for the base case we have:

For $T(0) = 0$

Then,

$k - 1$ *is less than k so from the above recurence relation we* n^2, *substitute* $k - 1$ *into that.*

$T(n) = (n) + T(n-k) + (k-1)^2$

Using iterative substitution:

$T(n) = (n) + T(n-k) + (k-1)^2$ **Iteration #1**

$T(n-k) = (n-k) + T(n-2k) + (k-1)^2$

$T(n) = n + (n-k) + T(n-2k) + (k-1)^2 + (k-1)^2$

$T(n) = 2n - k + 2(k-1)^2 + T(n-2k)$ **Iteration #2**

$T(n-2k) = (n-2k) + T(n-3k) + (k-1)^2$

$T(n) = 2n - k + (n-2k) + T(n-3k) + 3(k-1)^2$

$T(n) = 3n - 3k + T(n-3k) + 3(k-1)^2$ **Iteration #3**

.

.

.

By rearranging we get:

$T(n) = 3n - 3(k-1)^2 + T(n-3k) - k(1 + 2 + 3 + 4....)$

n is a multiple of k, so to reach our base case of T(0) = 0 we need $\frac{n}{k}$ iterations.

Then

$T(n) = (\frac{n}{k})3n - 3(\frac{n}{k})(k-1)^2 + (\frac{n}{k})T(n-3k) - (\frac{n}{k})k(1 + 2 + 3....)$

Part of T(n) calculation:

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

The last term can be written as a sum

$$T(n) = (\frac{3n^2}{k}) + 3n(k) - 2(3n) + (\frac{3n}{k}) + (\frac{n}{k})T(n-3k) - k(\sum_{i=1}^{\frac{n}{k}} i)$$

The 3 is just a constant so if we take it out and also simplify the summation using arithmetic series summation simplification:

$$T(n) \leq c * ((\frac{n^2}{k}) + 3n(k) - 2(3n) + (\frac{3n}{k}) + 0 - k(\frac{n}{k}\frac{(\frac{n}{k}+1)}{2}))$$

$$T(n) \leq c * ((\frac{n^2}{k}) + n(k) - 2(n) + (\frac{n}{k}) - (\frac{1}{2}(\frac{n^2}{k} + n)))$$

From this we see that the biggest term is forming an upper bound for some constant c $\geq$ 3.

Thus,

$$T(n) = O(c * (\frac{n^2}{k} + n(k) - 2(n) + (\frac{n}{k}) - (\frac{1}{2}(\frac{n^2}{k} + n))))$$

c)

The worst case of quick sort occurs in these situations:
Array is already sorted in the correct way.
Array is already sorted in the reverse way.
All elements are the same value.
And from the lecture notes we know the complexity to be $\Theta(n^2)$.
From the lecture notes we know that in these cases the partition size is usually equal to a size of 1. We also see from the lecture and from part B that k is the partition size as the quickSort breaks the array in n-k.
So for the worst case k = 1. By substituting the partition of k = 1 into the equation we get:

$$\begin{aligned} T(n) &= O(c * (\tfrac{n^2}{k} + n(k) - 2(n) + (\tfrac{n}{k}) - (\tfrac{1}{2}(\tfrac{n^2}{k} + n)))) \\ &= (\tfrac{n^2}{1} + n(1) - 2(n) + (\tfrac{n}{1}) - (\tfrac{1}{2}(\tfrac{n^2}{1} + n))) \\ &= (n^2 + n(1) - 2(n) + n - (\tfrac{1}{2}(n^2 + n)) \\ &= O(\tfrac{n^2}{1} + n(1) - 2(n) + (\tfrac{n}{1}) - (\tfrac{1}{2}(\tfrac{n^2}{1} + n)))) \end{aligned}$$

Then for a precise complexity we get

$$= \Theta(n^2)$$

Likewise KQuickSort has the same worst case when insertion sort is used at n < k.

4) Prove that a tree with n vertices has n - 1 edges.

Proof by induction:
Base Case:
When n = 1 vertex.
There are 1 - 1 = 0 edges. A single vertex has no edges as there are no other vertices to connect to.
When n = 2 vertices.
There are 2 - 1 = 1 edges.
When n = 3 vertices.
There are 3 - 1 = 2 edges.
So the base case holds.
Inductive Hypothesis:
Assume this statement holds for any tree with n vertices where n > 1.
Inductive Step:
By the definition a tree is an undirected graph. A tree denoted as T with n vertices, for n > 1 has atleast one edge, meaning the number of edges can not be zero.
Let c be a connecting edge that connects two vertices A and B. Since T is a tree and by definition only has one path that connects the two vertice. If we delete the edge c, then the tree is split into two sub-trees, T_1 & T_2. Respectively T_1 & T_2 have n_1 & n_2 vertices where $n_1 < n$ & $n_2 < n$, $n_1 + n_2 = n$ *total vertices*.

3 KQuickSort 2.5 / 5

+ **5 pts** Correct

✓ **+ 1.5 pts** **part a: recurrence is correct**

+ **1 pts** part a partially : a mistake or missing

+ **0.5 pts** part a: mostly incorrect, some correct parts

+ **0 pts** part a: recurrence completely incorrect or missing

+ **0.5 pts** part b: Usage of constants during substitution and in recurrence

✓ **+ 0 pts** **part b: incorrect usage or missing using some constants during substitutions and in recurrence**

+ **0.5 pts** part b: correct Substitution of T(k-1)

✓ **+ 0 pts** **part b: Incorrect or incomplete substitution of T(k-1)**

+ **1 pts** part b: Correct Substitution of T(n-k)

+ **0.5 pts** part b: Partially correct Substitution of T(n-k)

✓ **+ 0 pts** **part b: Incorrect or missing substitution of T(n-k)**

+ **0.5 pts** part b: Correct recurrence solution

✓ **+ 0 pts** **part b: Incorrect or missing final recurrence Solution**

✓ **+ 0.5 pts** **part b: Correct asymptotic notation O(n^2) based on the recurrence solution**

+ **0 pts** part b: Incorrect or missing Asymptotic Notation or incorrect related arguments or not based on the recurrence solution

✓ **+ 0.5 pts** **part c: Correct**

- **0.5 pts** Miscellaneous: minor mistakes or missing

+ **0 pts** part b: not submitted or totally wrong

+ **0 pts** part c: completely incorrect or missing or based on incorrect arguments from previous section or not base on part b

+ **0 pts** Not submitted or totally wrong

7 cannot ignore constant at this point

8 3?

9 ?

Thus,

$$T(n) = O(c * (\frac{n^2}{k} + n(k) - 2(n) + (\frac{n}{k}) - (\frac{1}{2}(\frac{n^2}{k} + n))))$$

c)

The worst case of quick sort occurs in these situations:
Array is already sorted in the correct way.
Array is already sorted in the reverse way.
All elements are the same value.
And from the lecture notes we know the complexity to be $\Theta(n^2)$.
From the lecture notes we know that in these cases the partition size is usually equal to a size of 1. We also see from the lecture and from part B that k is the partition size as the quickSort breaks the array in n-k.
So for the worst case k = 1. By substituting the partition of k = 1 into the equation we get:

$$\begin{aligned} T(n) &= O(c * (\tfrac{n^2}{k} + n(k) - 2(n) + (\tfrac{n}{k}) - (\tfrac{1}{2}(\tfrac{n^2}{k} + n)))) \\ &= (\tfrac{n^2}{1} + n(1) - 2(n) + (\tfrac{n}{1}) - (\tfrac{1}{2}(\tfrac{n^2}{1} + n))) \\ &= (n^2 + n(1) - 2(n) + n - (\tfrac{1}{2}(n^2 + n)) \\ &= O(\tfrac{n^2}{1} + n(1) - 2(n) + (\tfrac{n}{1}) - (\tfrac{1}{2}(\tfrac{n^2}{1} + n)))) \end{aligned}$$

Then for a precise complexity we get

$$= \Theta(n^2)$$

Likewise KQuickSort has the same worst case when insertion sort is used at n < k.

4) Prove that a tree with n vertices has n - 1 edges.

Proof by induction:
Base Case:
When n = 1 vertex.
There are 1 - 1 = 0 edges. A single vertex has no edges as there are no other vertices to connect to.
When n = 2 vertices.
There are 2 - 1 = 1 edges.
When n = 3 vertices.
There are 3 - 1 = 2 edges.
So the base case holds.
Inductive Hypothesis:
Assume this statement holds for any tree with n vertices where n > 1.
Inductive Step:
By the definition a tree is an undirected graph. A tree denoted as T with n vertices, for n > 1 has atleast one edge, meaning the number of edges can not be zero.
Let c be a connecting edge that connects two vertices A and B. Since T is a tree and by definition only has one path that connects the two vertice. If we delete the edge c, then the tree is split into two sub-trees, T_1 & T_2. Respectively T_1 & T_2 have n_1 & n_2 vertices where $n_1 < n$ & $n_2 < n$, $n_1 + n_2 = n$ *total vertices*.

So the total number of edges by using our assumption:

$\#\ of\ edges\ in\ T\ =\ (n_1 - 1)\ +(n_2 - 1)\ +1,\ \ +1\ edge\ from\ addition\ of\ two\ sub\ trees$

$\#\ of\ edges\ in\ T\ =\ (n_1 + n_2)\ -2+1$

$\#\ of\ edges\ in\ T\ =\ (n_1 + n_2)\ -1$

$(n_1 + n_2)$ is equivalent to the whole Tree.

So,

$\#\ of\ edges\ = n\ -1$

Thus the inductive step holds.

4 Number of Tree Edges 2.5 / 3

+ **3 pts** Correct

✓ **+ 0.5 pts** **Base case correct**

+ **0 pts** Base case incorrect or missing

+ **0.5 pts** Correct Inductive hypothesis

✓ **+ 0 pts** **Incorrect or missing inductive hypothesis**

✓ **+ 2 pts** **Inductive step correct**

+ **1 pts** Inductive step partially correct

+ **0.5 pts** Inductive step : not convincing and mostly incorrect

+ **0 pts** Inductive Step totally incorrect or missing

- **0.5 pts** A minor mistake in Inductive step

+ **0 pts** Totally Incorrect or not submitted

+ **0 pts** Very minor mistakes (no points deducted)

10 we cannot conclude this from the I.H.

11 the same.