CSCI-4470 HW-2

Due: Sep 22 2023

Homework 2

Jun Wang
ID: 811574679

Upload a soft copy of your answers (single pdf file) to the submission link on elc before the due date. The answers to the homework assignment should be your own individual work. Make sure to show all the work/steps in your answer to get full credit. Answers without steps or explanation will be given zero.

Extra credit: There is 5 percentage extra credit if you don't submit hand written homework including the figures. You can use latex or any other tool to write your homework. For figures you can use any drawing tool and include the figure as a jpeg or a png file in your latex file.

---

1. (10 points) Using Figure 7.1 as a model, illustrate the operation of function PARTITION on the array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$ **.**

i	1	2	3	4	5	6	7	8	9	10	11	12
	p, j, i											r
(1)	**13**	19	9	5	12	8	7	4	21	2	6	**11**
	p, i	j										r
(2)	13	**19**	9	5	12	8	7	4	21	2	6	**11**
	p, i		j									r
(3)	13	19	**9**	5	12	8	7	4	21	2	6	**11**
	p	i		j								r
(4)	`9`	19	`13`	**5**	12	8	7	4	21	2	6	**11**
	p		i		j							r
(5)	9	`5`	13	`19`	**12**	8	7	4	21	2	6	**11**
	p		i			j						r

i	1	2	3	4	5	6	7	8	9	10	11	12
(6)	9	5	13	19	12	**8**	7	4	21	2	6	**11**
	p			i			j					r
(7)	9	5	`8`	19	12	`13`	**7**	4	21	2	6	**11**
	p				i			j				r
(8)	9	5	8	`7`	12	13	`19`	**4**	21	2	6	**11**
	p					i			j			r
(9)	9	5	8	7	`4`	13	19	`12`	**21**	2	6	**11**
	p					i				j		r
(10)	9	5	8	7	4	13	19	12	21	**2**	6	**11**
	p						i				j	r
(11)	9	5	8	7	4	`2`	19	12	21	`13`	**6**	**11**
	p							i				r
(12)	9	5	8	7	4	2	`6`	12	21	13	`19`	11
	p											r
(13)	9	5	8	7	4	2	6	`11`	21	13	19	`12`

```
Partition(A,p,r)        // p is the starting index of the array,
                        // r is the ending index of the array
1. x = A[r]         // x is the pivot element, which is the last element of the array
2. i = p - 1        // i is initialized to p - 1
3. for j = p to r-1     // j starts from p
4.      if A[j]<= x
5.          i = i + 1
6.          exchange A[i] with A[j]
            // For each j from p to r-1, if A[j] ≤ x,
            // increment i and exchange A[i] with A[j].
7. exchange A[i+1] with A[r]       // Exchange A[i+1] with A[r] (the pivot).
8. return i+1
```

---

2. (15 points) In class we have shown that worst case runtime for quick-sort is $O(n^2)$ using similar idea show that quicksort's best-case running time is $\Omega(n\log(n))$.

Given Recurrence Relation:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(q-1) + T(n-q) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Step 1: Assumption

- Assume that, $T(n) \geq c \cdot n \log(n)$

Using our assumption:

- $T(q-1) \geq c \cdot (q-1) \log(q-1)$
- $T(n-q) \geq c \cdot (n-q) \log(n-q)$

Substitute these into the recurrence:

- $T(n) \geq c \cdot (q-1) \log(q-1) + c \cdot (n-q) \log(n-q) + kn$, k is a constant from the $\Theta(n)$ term.

Step 2: Differentiate to find the Minimum for best case scenario with respect to q. **(First derivative)**

- $\frac{d}{dq} c \cdot (q-1) \log(q-1) + \frac{d}{dq} \left[c \cdot (n-q) \log(n-q) \right] + \frac{d}{dq} [kn] = 0$

Differentiate the term $c \cdot (q-1) \log(q-1)$ with respect to q:

Apple the product rule, $u = (q-1)$, $v = \log(q-1)$, then $\frac{d}{dq} uv = u'v + uv'$

- $u' = \frac{d}{dq}(q-1) = 1$
- By the chain rule, $v' = \frac{1}{q-1} \cdot \frac{d}{dq} \log(q-1) = \frac{1}{q-1} \cdot 1 = \frac{1}{q-1}$

Plugging in:

- $\frac{d}{dq} c \cdot (q-1) \log(q-1) = c \left[\log(q-1) + (q-1) \cdot \frac{1}{q-1} \right] = c \left[\log(q-1) + 1 \right]$

Differentiate the term $c \cdot (n-q) \log(n-q)$ with respect to q

Apple the product rule, $u = (n-q)$, $v = \log(n-q)$, then $\frac{d}{dq} uv = u'v + uv'$

- $u' = \frac{d}{dq}(n-q) = -1$
- By the chain rule, $v' = \frac{1}{n-q} \cdot \frac{d}{dq} \log(n-q) = \frac{1}{n-q} \cdot (-1) = \frac{-1}{n-q}$

Plugging in:

- $\frac{d}{dq} c \cdot (n-q) \log(n-q) = c \left[-\log(n-q) - (n-q) \cdot \frac{-1}{n-q} \right] = c \left[-\log(n-q) - 1 \right]$

Differentiate the term kn with respect to q:

- $\frac{d}{dq} kn = 0$

Combine the terms: which is First Derivative

- $\frac{d}{dq} \left[c \cdot (q-1) \log(q-1) + c \cdot (n-q) \log(n-q) + 0 \right] = 0$
- $= c \left[\log(q-1) + 1 \right] + c \left[-\log(n-q) - 1 \right] = 0$,
- $= c \left[\log(q-1) + 1 \right] - c \left[\log(n-q) + 1 \right] = 0$ **First Derivative**

- $= c\left[\log(q-1) - \log(n-q)\right] = 0$, Since c is a constant, and won't affect the minimization, c can be ignored for finding q

Step 3: Setting this to 0 gives to find q:

- $\log(q-1) - \log(n-q) = 0$
- By the Logarithm properties, $\log(q-1) = \log(n-q) = q - 1 = n - q$
- $2q = n + 1$
- $q = \frac{n+1}{2}$, the value of q that minimizes $f(q)$ and thus represents the best-case scenario.

Step 4: To verify that $q = \frac{n+1}{2}$ is a minimum, To find the second derivative, $f''(q)$ by differentiating each term of $f'(q)$ with respect to q.

- $f'(q) = c\left[\log(q-1) + 1\right] - c\left[\log(n-q) + 1\right]$

Differentiate the term $c\left[\log(q-1) + 1\right]$ with respect to q.

- Apply the chain rule, $\frac{d}{dq}\log(q-1) = \frac{1}{q-1}$, and $\frac{d}{dq}1 = 0$
- $c \cdot \frac{d^2}{dq^2}\left[\log(q-1) + 1\right] = c \cdot \frac{1}{q-1}$

Differentiate the term $-c\left[\log(n-q) + 1\right]$ with respect to q.

- Apply the chain rule, $\frac{d}{dq}\log(n-q) = \frac{-1}{n-q}$, and $\frac{d}{dq}1 = 0$
- $-c \cdot \frac{d^2}{dq^2}\left[\log(n-q) + 1\right] = -c \cdot \frac{1}{n-q}$

Combine the terms, and apply $q = \frac{n+1}{2}$

- $f''(q) = c\left[\frac{1}{q-1} - \frac{1}{n-q}\right]$
- $f''(\frac{n+1}{2}) = c\left[\frac{1}{q-1} - \frac{1}{n-q}\right]$

Substitute q with $\frac{n+1}{2}$ in the first term

- $\frac{1}{q-1} \to \frac{1}{\frac{n+1}{2}-1} = \frac{1}{\frac{n-1}{2}} = \frac{2}{n-1}$

Substitute q with $\frac{n+1}{2}$ in the second term

- $\frac{1}{n-q} \to \frac{1}{n-\frac{n+1}{2}} = \frac{1}{\frac{n-1}{2}} = \frac{2}{n-1}$

Combine both terms, $n \geq 2$

- $f''\left(\frac{n+1}{2}\right) = c\left[\frac{2}{n-1} - \frac{2}{n-1}\right]$
- $f''\left(\frac{n+1}{2}\right) = 0$, f'' at $q = \frac{n+1}{2}$ is zero, indicating a potential minimum.
- Thus, the best-case running time of quicksort is $\Omega(n \log n)$.

---

3. (10 points) So far we have discussed four sorting algorithms in the course merge sort, insertion sort, heap sort and quick sort. Consider an array $A = \langle 1, 3, 5, 10, 7, 15, 21, 14, 27, 32, 39, 41 \rangle$ and answer the questions below.

- **(a)** What do you expect from the performances of these algorithms on the above data set. You can talk about if algorithm is going to do more comparisons or less comparisons in this situation compare to normal case and then give a big O estimate of all the algorithms.

Algorithm Name	Best-case	Average-case	Worst-case	Memory	Stable
MergeSort	$n\log(n)$	$n\log(n)$	$n\log(n)$	worst: n	yes
HeapSort	$n\log(n)$	$n\log(n)$	$n\log(n)$	1	no
InsertionSort	n	n^2	n^2	1	yes
QuickSort	$n\log(n)$	$n\log(n)$	n^2	average: $\log(n)$ worst: n	no
Bubblesort	n	n^2	n^2	1	yes
SelectionSort	n^2	n^2	n^2	1	no

    - The array is mostly sorted. Based on the table
    1. **Merge Sort**: Consistent $O(n\log n)$ performance.
    2. **Insertion Sort**: Efficient on nearly sorted data, close to $O(n)$.
    3. **Heap Sort**: Always $O(n\log n)$, unaffected by data order.
    4. **Quick Sort**: Performance depends on pivot choice, average $O(n\log n)$.
- **(b)** Based on the answer in part a which algorithm is most suitable to use in the given situation.
    - **Insertion Sort** is most efficient for this dataset.

---

4. (15 points) Consider the problem to search a list of n elements for a key. Use the decision tree method to prove that, regardless if the input list is sorted or not, this search problem has time complexity lower bound $\Omega(\log(n))$.

- By Using the decision tree method, each comparison made during the search can be represented as an internal node, and the possible outcomes as branches. The tree's height represents the maximum comparisons needed. For a list of n elements, the tree has at least n leaves (each representing a potential outcome). A binary tree with n leaves has a height of at least $\log(n)$. Thus, the minimum comparisons required, in the worst case, is $\Omega(\log(n))$, irrespective of the list being sorted or not.

---

5. (10 points) Using Figure 8.2 as a model, illustrate the operation of COUNTING-SORT on the array $A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2\rangle$.

- The original Array {6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2}
    - The min value is 0, The max value is 6
- The initialized counting Array C={0,0,0,0,0,0,0}
    - the size of array C is $k + 1 = 6 + 1 = 7$

Input Array	6	0	2	0	1	3	4	6	1	3	2
Index[i]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
Counting Array c	2	2	2	2	1	0	2				
Index[i] of C	C[0]	C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	x	x	x	x
Cumulated C	2	4	6	8	9	9	11				
Decrease by 1	2->1	3->2	5->4	7->6	8	9	10->9				

$C[0] = 2$
$C[1] = C[1] + C[0] = 2 + 2 = 4$, $C[2] = C[2] + C[1] = 2 + 4 = 6$,
$C[3] = C[3] + C[2] = 2 + 6 = 8$, $C[4] = C[4] + C[3] = 1 + 8 = 9$,
$C[5] = C[5] + C[4] = 0 + 9 = 9$, $C[6] = C[6] + C[5] = 2 + 9 = 11$

Sorted Array B	0	0	1	1	2	2	3	3	4	6	6
Index[i] of B	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]	[10]

$A[10] = 2$, and $C[2] = 6$, then $B[6 - 1] = 2$, $C[2] = 6 - 1 = 5$
$A[9] = 3$, and $C[3] = 8$, then $B[8 - 1] = 3$, $C[3] = 8 - 1 = 7$
$A[8] = 1$, and $C[1] = 4$, then $B[4 - 1] = 1$, $C[1] = 4 - 1 = 3$
$A[7] = 6$, and $C[6] = 11$, then $B[11 - 1] = 6$, $C[6] = 11 - 1 = 10$
$A[6] = 4$, and $C[4] = 9$, then $B[9 - 1] = 4$, $C[4] = 9 - 1 = 8$
$A[5] = 3$, and $C[3] = 7$, then $B[7 - 1] = 3$, $C[3] = 7 - 1 = 6$
$A[4] = 1$, and $C[1] = 3$, then $B[3 - 1] = 1$, $C[1] = 3 - 1 = 2$
$A[3] = 0$, and $C[0] = 2$, then $B[2 - 1] = 0$, $C[0] = 2 - 1 = 1$
$A[2] = 2$, and $C[2] = 5$, then $B[5 - 1] = 2$, $C[2] = 5 - 1 = 4$
$A[1] = 0$, and $C[0] = 1$, then $B[1 - 1] = 0$, $C[0] = 5 - 1 = 4$
$A[0] = 6$, and $C[6] = 10$, then $B[10 - 1] = 6$, $C[6] = 10 - 1 = 9$

---

6. (10 points) Using Figure 8.3 as a model, illustrate the operation of RADIX-SORT on the following list of English words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

- - -	- - ↓	- ↓ -	↓ - -
COW	SEA	TAB	BAR
DOG	TEA	BAR	BIG
SEA	MOB	EAR	BOX

- - -	- - ↓	- ↓ -	↓ - -
RUG	TAB	TAR	COW
ROW	DOG	SEA	DIG
MOB	RUG	TEA	DOG
BOX	DIG	DIG	EAR
TAB	BIG	BIG	FOX
BAR	BAR	MOB	MOB
EAR	EAR	DOG	NOW
TAR	TAR	COW	ROW
DIG	COW	ROW	RUG
BIG	ROW	NOW	SEA
TEA	NOW	BOX	TAB
NOW	BOX	FOX	TAR
FOX	FOX	RUG	TEA

---

7. (10 points each) In the algorithm SELECT (using median of median as a pivot value) discussed in the class we created 5 groups and showed that you can make i^{th} selection in $O(n)$ in the worst case.

```
SELECT(A, p, r, i)
1. while(r - p + 1) mod 5 ≠ 0
2. for j = p + 1 to r      // put the minimum into Alp]
3.    if A[p] > A[j]
4.        exchange A[p] with A[j]
5. // If we want the minimum of A[p: r], we're done.
6. if i == 1
7.    return A[p]
8.  // Otherwise, we want the (i - 1)st element of A[p + 1: r].
9. p = p + 1
10. i = i - 1
11. g = (r - p + 1)/5      // number of 5-element groups
12. for j = p to p + g - 1    // sort each group
13.     sort<A[j], A[j + g], A[j + 2g], A[j + 3g], A[j + 4g]> in place
14.   // All group medians now lie in the middle fifth of A[p: r].
15.   // Find the pivot x recursively as the median of the group medians.
16. x = SELECT(A, p + 2g, p + 3g - 1, ⌈g/2⌉)
17. q = PARTITION-AROUND(A, p, r, x)    // partition around the pivot
18. // The rest is just like lines 3-9 of RANDOMIZED-SELECT.
19. k = q - p + 1
20. if i == k
21.     return A[q]     // the pivot value is the answer
22. elseif i < k
23. return SELECT(A, p, q - 1, i)
24. else return SELECT(A, q + 1, r, i - k)
```

(a) What will happen to the complexity of SELECT algorithm (discussed in Chapter 9) if we create 9 groups instead of 5? Show all your work/steps.

Initial Array A = {10, 38, 87, 55, 47, 5, 3, 9, 12, 88, 19, 22, 53, 98, 17, 37, 35, 63, 72, 33, 93, 87, 15, 66}

G1	G2	G3	G4	G5	G6	G7	G8	G9
10	55	3	88	53	37	72	93	15
38	47	9	19	98	35	33	87	66
87	5	12	22	17	63			

1. The Recurrence Relation for 9 Groups, the recurrence relation for the 9-group division

- $T(n) \leq T\left(\left\lceil \frac{n}{9} \right\rceil\right) + T\left(\left(\frac{8n}{9}\right) + 6\right) + O(n)$

2. Solving the Recurrence

Step 1: prove $T(n) = O(n)$ using the substitution method.

- Assume $T(n) \leq cn$, aiming to find c and a.

Step 2: Substitute $T(n) \leq cn$ into the recurrence:

- $T(n) \leq c\left(\frac{n}{9} + 1\right) + c\left(\left(\frac{8n}{9}\right) + 6\right) + an$

Distribute c in the term $c\left(\frac{n}{9} + 1\right)$ and $c\left(\left(\frac{8n}{9}\right) + 6\right)$

- $c\left(\frac{n}{9}\right) + c$ and $c\left(\left(\frac{8n}{9}\right)\right) + 6c$
- $T(n) \leq c\left(\frac{n}{9}\right) + c + c\left(\left(\frac{8n}{9}\right)\right) + 6c + an$

Combine the n terms by adding $\frac{n}{9}c$ and $\left(\frac{8n}{9}\right)c$

- $T(n) \leq c\left(\frac{n}{9} + \left(\frac{8n}{9}\right)\right) + 7c + an$
- $T(n) \leq cn + 7c + an$

Simplify, and satisfy $T(n) \leq cn$:

- $T(n) \leq cn + 7c + an$
- $cn + 7c + an \leq cn$

Bring the term cn to the right to have it with cn:

- $7c + an \leq 0$
- $an \leq -7c$
- $a \leq -7$
- $a > -7$
- $n \cdot a \leq -7c$
- $n \geq \frac{-7c}{a}$
- $n > \frac{-7c}{a}$

The 9-group division results in a linear-time selection algorithm, similar to the 5-group division.
The partitioning is more balanced in the 9-group division, making it potentially more efficient in practice.

(b) Show that if you create 3 groups then algorithm will no longer be linear.

- Suppose now that we use groups of size 3 instead. the possible recurrence will be $T(n) = T(\lceil \frac{n}{3} \rceil) + T(\frac{4n}{6}) + O(n) \geq T(\frac{n}{3}) + T(\frac{2n}{3}) + O(n)$. It could be $\geq cn\log(n)$.
- Assume that $T(m) \geq c(\frac{m}{3}) \cdot \log(\frac{m}{3}) + c(\frac{2m}{3}) \cdot \log(\frac{2m}{3}) + O(m) \geq cm \cdot \log(m) + O(m)$. it grows more quickly than linear.

---

8. The version of PARTITION given in Chapter 7 is not the original partitioning algorithm. Here is the original partitioning algorithm, which is due to C. A. R. Hoare.

```
QUICKSORT-HOARE(A, p, r) {
    if (p < r) {
        q = partitionHoare(arr, left, right);
        QUICKSORT-HOARE(arr, p, q);
        QUICKSORT-HOARE(arr, q + 1, r);
    }
}
```

```
PARTITION(A, p, r)
1 x = A[r]      // the pivot
2 i = p - 1     // highest index into the low side
3 for i = p to r - 1        // process each element other than the pivot
4   if A[j] ≤ x     // does this element belong on the low side
5       i = i + 1       // index of a new slot in the low side
6       exchange A[i] with A[j]     // put this element there
7 exchange A[i + 1] with A[r]        // pivot goes just to the right of the low side
8 return i + 1      // new index of the pivot
```

```
HOARE-PARTITION(A, p, r)
1 x = A[p]  // pivot `x` is chosen as the first element of the segment.
2 i = p - 1
3 j = r + 1
4 while TRUE
5   repeat
6       j = j - 1
7   until A[j] ≤ x
8   repeat
9       i = i + 1
10  until A[i] ≥ x
11  if i < j
12      exchange A[i] with A[j]
13  else return j
```

[a] (10 points) Demonstrate the operation of `HOARE-PARTITION` on the array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 \rangle$, showing the values of the array and the indices `i` and `j` after each iteration of the while loop in lines 4-13.

- $p = 1, r = |A| = 12$, pivot $x = 13, , i = 0, j = 13$

i	1	2	3	4	5	6	7	8	9	10	11	12	-
i=-1	p											r	j=13
	13	19	9	5	12	8	7	4	11	2	6	**21**	
	p, i=1										j = 11	r	
	`6`	19	9	5	12	8	7	4	11	2	`13`	**21**	
	p	i=2								j=10		r	
	6	`13`	9	5	12	8	7	4	11	2	`19`	**21**	
	p	j=2								i=10			
	6	`13`	9	5	12	8	7	4	11	2	`19`	**21**	

---

[b] (10 points) Looking at the result in **part [a]** and the return value by the partition function comment how `HOARE-PARTITION` is different from the `PARTITION` procedure in chapter-7.

- Pivot Selection different, for `PARTITION` function `A[r]`, for `HOARE-PARTITION` function `A[p]`
- Index `i`, and `j`, `PARTITION` only has `i` move from the start to the end of the array, `HOARE-PARTITION` have both `i` and `j` moved during iterations

- Return value, `PARTITION` returns the index of the pivot after placed in its correct position
- `HOARE-PARTITION` returns the index `j`, which is not necessarily the correct position, or final position of pivot

---

[c] (10 points) Now compare the results of `PARTITION` procedure in Chapter 7 with `HOARE-PARTITION` when all elements in `A[p : r]` are equal. Describe a practical advantage of `HOARE-PARTITION` over `PARTITION` for use in quicksort.

For both functions, When all elements in `A[p : r]` are equal:

- **PARTITION**: Places almost all elements to the left of the pivot, leading to an imbalanced split, which is worst case of quicksort algorithm.
- **HOARE-PARTITION**: Splits the array more evenly, as both `i` and `j` move towards each other.
- **Advantage**: `HOARE-PARTITION` provides a balanced partition, making subsequent quicksort calls more efficient. In contrast, `PARTITION` can degrade quicksort's performance to $O(n^2)$ in this scenario.

---

[d] (5 points each) Assuming that the subarray `A[p : r]` contains at least two elements, prove the following for `HOARE-PARTITION` algorithm.

i. The indices i and j are such that the procedure never accesses an element of `A` outside the subarray `A[p : r]`.

- **Initialization:** $i = p - 1$ and $j = r + 1$, **Index j:** j decreases until $A[j] \leq x$. It won't go beyond p, **Index i:** i increases until $A[i] \geq x$. It won't exceed r. **Therefore:** i and j stay within $A[p : r]$.

ii. When `HOARE-PARTITION` terminates, it returns a value j such that $p \leq j < r$.

- **Termination:** Ends when $i \geq j$, **Return Value:** Returns j. Since i is right of j and at most r, $j < r$. And $j \geq p$ as it moves left. At termination, $p \leq j < r$.
