Assignment 2

Name: Mousumi Akter
SID: 904095347
I Discussed with: Alex Knipper and Md Mahadi Hassan

February 28, 2021

1. What is the best estimate (tightest upper bound) on the number of times the while loop of power-iterative will execute?

```
# Pseudocode for Power(x, n)
Power(x, n):
      # Initialization
      result = 1
      while n > 0 :
            # If n is odd, multiply result with x
            if odd(n):
               result = result*x
            n = floor(n/2)
            # Change x to x^2
            x =x*x
      return result
```

   If n=4, while loop will be executed 2 times for changing x to x^2 and then later change that to x^4 and 1 times for changing the result.

   If n=8, while loop will be executed 3 times for changing x and 1 times for changing the result.

   Thus while loop will be executed $\log_2 n + 1$ times.

   What is the approximate complexity of power-iterative?

   Time Complexity of initialization is O(1) and time complexity of while loop is $O(\log_2 n)$. Thus the approximate complexity of power-iterative is $O(\log_2 n)$.

2. Read the MERGE algorithm on pp.31 of CLRS book for this problem (also shown as follows). Calculate the complexity of the Merge algorithm

using (1) the approximate method and (2) the detailed method. You may assume that n_1=n_2=n/2. Fill in the table below appropriately. Note: step 3 of the algorithm is not executable.

```
MERGE(A, p, q, r)
 1  n_1 = q - p + 1
 2  n_2 = r - q
 3  let L[1..n_1 + 1] and R[1..n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i - 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

Approximate analysis:

Step #	Complexity stated as O(_)
1	O(1)
2	O(1)
4	Complexity of # of executions: $O(n_1)$ or O(n/2)
5	O(1)
Loop 4-5	Complexity of entire loop: $O(n_1)$ or O(n/2)
6	Complexity of # of executions: $O(n_2)$ or O(n/2)
7	O(1)
Loop 6-7	Complexity of entire loop: $O(n_2)$ or O(n/2)
8	O(1)
9	O(1)
10	O(1)
11	O(1)
12	Complexity of # of executions: O(r-p+1) or O(r-p)
13	O(1)
14	O(1)
15	O(1)
16	O(1)
17	O(1)
13-17	Complexity of single execution of loop body: O(1)
12-17	Complexity of entire loop: O(r-p)
1-17	Complexity of algorithm:O(n/2) or O(n)

Detailed analysis:

Step #	Cost of single execution	# of times executed
1	$c_1 = 3$	1
2	$c_2 = 2$	1
4	$c_4 = 1$	$(n_1 + 1)$
5	$c_5 = 4$	n_1
6	$c_6 = 1$	$(n_2 + 1)$
7	$c_7 = 3$	n_2
8	$c_8 = 1$	1
9	$c_9 = 1$	1
10	$c_{10} = 1$	1
11	$c_{11} = 1$	1
12	$c_{12} = 1$	$(r - p + 2)$
13	$c_{13} = 1$	$(r - p + 1)$
14	$c_{14} = 1$	$(r - p + 1)$
15	$c_{15} = 1$	$(r - p + 1)$
16	$c_{16} = 1$	$(r - p + 1)$
17	$c_{17} = 1$	$(r - p + 1)$

$$\mathbf{T(n)} = c_1 + c_2 + c_4 (n_1 + 1) + c_5 n_1 + c_6(n_2 + 1) + c_7 n_2 + c_8 + c_9 + c_{10} + c_{11} + c_{12}(r - p + 2) + (c_{13} + c_{14} + c_{15} + c_{16} + c_{17})(r - p + 1)$$

$$= (c_1 + c_2 + c_4 + c_6 + c_8 + c_9 + c_{10} + c_{11} + 2c_{12} + c_{13} + c_{14} + c_{15} + c_{16} + c_{17}) + (c_4 + c_5 + c_6 + c_7)(n/2) + (c_{12} + c_{13} + c_{14} + c_{15} + c_{16} + c_{17})(r - p)$$

$$= m_0 + m_1 n + m_2(r - p)$$

Assuming,

$(c_1 + c_2 + c_4 + c_6 + c_8 + c_9 + c_{10} + c_{11} + 2c_{12} + c_{13} + c_{14} + c_{15} + c_{16} + c_{17}) = m_0$

$(c_4 + c_5 + c_6 + c_7)/2 = m_1$

$c_{12} + c_{13} + c_{14} + c_{15} + c_{16} + c_{17} = m_2$

Since r-p is always less than n, T(n) = O(n)

3. Use the Recursion Tree Method to show that $T(n) = 2(2^n) - 1$ for a recursive algorithm characterized by the recurrences $T(n) = 2T(n-1) + 1; T(0) = 1$. You must fill in the table below for the first three levels of the Recursion Tree and for the base case level and the level above. Then write out the expression $T(n)$ that you get by adding all values in the last column and simplifying using results from Appendix A to show the above. This simplification must be shown to get any credit.Finally, state the most accurate complexity order of this algorithm:

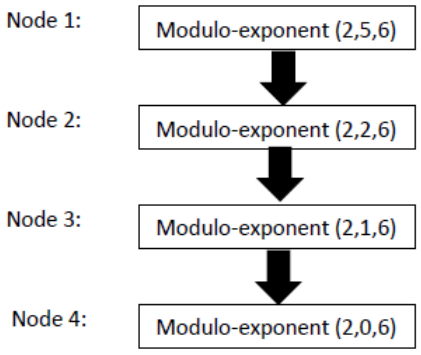
Level #	# of recursive executions at this level as a function of level #	Input size to each execution	Additional work done by each execution	Total work done at this level as a function of level #
1	0	n	1	2^0
2	1	n-1	1	2^1
3	2	n-2	1	2^2
4	4	n-3	1	2^4
	.	.	.	.
	.	.	.	.
	.	.	.	.
n	2^n	0	1	2^n

T(n) = $1+2+4+8+....+2^n = \frac{2^{n+1}-1}{2-1} = 2^{n+1} - 1 = 2(2^n) - 1$. Hence proved.

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

4. Understand this recursive algorithm for computing (bn mod m). Then draw the Recursion Tree of it computing (2^5 mod 6). Note: At each node of the tree you should show the inputs and outputs of that execution clearly to receive credit.

```
Modulo-exponent(b,n,m: integers such that m≥2, n≥0)
1. if n==0 then return 1
2. else
3.       temp = Modulo-exponent(b,floor(n/2),m)
4.       temp = temp*temp
5.       if n is even then
                  return temp mod m
6.       else
7.                return (temp mod m)*(b mod m) mod m
```



The table mentioned below describes the input and output for each node:

Node	Input	Output
4	b = 2, n = 0, m = 6	Value returned is 1 because n = 0.
3	b = 2, n = 1, m = 6	Value returned by node 4 =1 ⇒ temp = 1 in line 3 ⇒ temp = 1 in line 4. Since n is odd, it returns (1 mod 6)*(2 mod 6) mod 6 = (1 * 2) mod 6 = 2
2	b = 2, n = 2, m = 6	Value returned by node 3 =2 ⇒ temp = 2 in line 3 ⇒ temp = 2*2 = 4 in line 4. Since n is even, it returns (4 mod 6) = 4
1	b = 2, n = 5, m = 6	Value returned by node 2 = 4 ⇒ temp = 4 in line 3 ⇒ temp = 4*4 = 16 in line 4. Since n is odd, it returns (16 mod 6)*(2 mod 6) mod 6 = (4 * 2) mod 6 = 8 mod 6 = 2

Therefore, the final result is 2.

5. Use the Detailed Method to determine the precise T(n) of the following iterative maximum subsequence sum (MSS) algorithm. You must show your work below to get any credit. The algorithm is as below.

Line #	Step	Single execution cost	# times executed
1	sum = max = 0	$c_1 = 3$	1
2	for i = p to q	$c_2 = 2$	q – p + 2
3	sum = 0	$c_3 = 1$	q – p + 1
4	for j = i to q	$c_4 = 1$	$\sum_{i=p}^{q}(q - i + 2)$
5	sum = sum + A[j]	$c_5 = 2$	$\sum_{i=p}^{q}(q - i + 1)$
6	if sum > max then	$c_6 = 1$	$\sum_{i=p}^{q}(q - i + 1)$
7	max = sum	$c_7 = 1$	$\sum_{i=p}^{q}(q - i + 1)$
8	return max	$c_8 = 1$	1

From this expression, we get T(n) by substituting p=1 and q=n.

$\therefore T(n) = c_1+(n-1+2)c_2+(n-1+1)c_3+(n^2-2n+5n+1-3+4)c_4/2+(n^2-2n+3n+1-1+2)(c_5+c_6+c_7)/2+1$

$= c_1+(n+1)c_2+nc_3+(n^2+3n+2)c_4/2+(n^2+n+2)(c_5+c_6+c_7)/2+1=k_2n^2+k_1n+k_0$ where k_0, k_1 and k_2 have been selected to replace the coefficients of n^2, n and constant respectively.

Thus, T(n)=O(n^2) for larger n.

6. Develop, state and solve the recurrence relations of the Recursive Divide & Conquer iterative algorithm as follows by answering the following questions.

```
MSS Algorithm-2 (A:array[p..q] of integer)
 1      if p=q then
 2              if A[p] > 0 then
 3                      return A[p]
 4              else return 0
 5      left-partial-sum = right-partial-sum = max-right = max-left = left-max-sum = right-max-sum = 0
 6      center = floor((p+q)/2)
 7      max-left = Algorithm-2(A[p..center])
 8      max-right = Algorithm-2(A[center+1..q])
 9      for i from center downto p do
10              left-partial-sum = left-partial-sum + A[i]
11              if left-partial-sum > left-max-sum then
12                      left-max-sum = left-partial-sum
13      for i from center+1 to q do
14              right- partial-sum = right-partial-sum + A[i]
15              if right- partial-sum > right-max-sum then
16                      right-max-sum = right- partial-sum
17      if max-left≤max-right then
18              if max-right≤left-max-sum+right-max-sum then
19                      return left-max-sum+right-max-sum
20              else return max-right
        else
21              if max-left<eft-max-sum+right-max-sum then
22                      return left-max-sum+right-max-sum
23              else return max-left
```

Step#	Single execution cost	# times executed
1	$c_1 = 1$	1
2	$c_2 = 2$	1
3	$c_3 = 2$	1
4	$c_4 = 1$	1
5	$c_5 = 6$	1
6	$c_6 = 4$	1
7	c_7	T(n/2)
8	c_8	T(n/2)
9	$c_9 = 2$	n/2
10	$c_{10} = 3$	n/2
11	$c_{11} = 1$	n/2
12	$c_{12} = 1$	n/2
13	$c_{13} = 2$	n/2
14	$c_{14} = 3$	n/2
15	$c_{15} = 1$	n/2
16	$c_{16} = 1$	n/2
17	$c_{17} = 2$	1
18	$c_{18} = 3$	1
19	$c_{19} = 2$	1
20	$c_{20} = 1$	1
21	$c_{21} = 2$	1
22	$c_{22} = 2$	1
23	$c_{23} = 1$	1

Which statements are executed when the input is a base case (provide line #s)? 1-4
What is the total cost of these? 1+2+2+1=6

Which statements are executed when the input is not a base case (provide line #s)? 1, 5-23 What is the total cost of these?

1+6+4+(2+3+1+1+2+3+1+1)(n/2)+2+3+2+1+2+2+1+2T(n/2) = 24 +7n+2T(n/2)

Provide the complete and precise two recurrence relations characterizing the complexity of MSS Algorithm-2:

T(n) = 6 when n=1

T(n) = 2T(n/2)+7n+24 when n>1

Now simplify the recurrence relations by:

1. If your recurrence relation for the non-base case input has multiple terms in it besides the term representing the recursive calls, keep only the largest n-term from them and drop the others; if your recurrence relation for the non-base case input has only one other term besides the term representing the recursive calls, keep it.

2. Take the largest numerical coefficient of all terms (excluding the term representing the recursive calls) in your two recurrence relations, round it up to the next integer if it is not an integer, and replace the numerical coefficients of all other terms (excluding the term representing the recursive calls) with this coefficient.

Provide the simplified recurrence relations below. T(n) = 6 when n=1

T(n) = 2T(n/2)+7n when n>1

Solve these recurrence relations using the Recursion Tree method, determine and state the T(n) of the algorithm. You must show your work below to get any credit.

Level #	# of recursive execution at this level as a function of level	Input size to each execution	Additional work done by	Total work done at this level
0	2^0	n	7n	7n
1	2^1	n/2	7n/2	7n
2	2^2	n/4	7n/4	7n
logn-1	$2^{logn-1} = n/2$	2	14	7n
log n	$2^{logn} = n$	1	7	7n

Total cost = 7n(logn+1)

7. Let Result [1..2] be an array of two integers. Modify steps of the Iterative MSS algorithm as in Question 3 to return the starting and ending indexes of the optimal (maximum) subsequence it found in its last line instead of the maximum sum value. Provide your modified algorithm below. Make only the minimum number of modifications necessary. You will need to add additional steps.

   Modified algorithm is below where modified parts have been shown in bold letters.

```
MSS Algorithm-1 (A:array[p..q] of integer)
        sum, max: integer
        Result: array[1..2] of integer
1       sum = max = 0
2       Result = <0, 0>
3       for i = p to q
4               sum = 0
5               for j = i to q
6                       sum = sum + A[j]
7                       if sum > max then
8                               max = sum
9                               Result = <i, j>
10      return Result
```

8. Suppose a recursive algorithm is characterized by these recurrences: $T(n) = 3T(n/2) + n; T(1) = 1$. Let a guessed solution be $T(n) = O(n^2)$. Prove that this guess is correct using the Substitution Method. State values of the constants n0 and c that you determine as part of the proof. You must clearly show the three parts of the inductive proof to get credit. Hints: (i) The goal of simplification in the Inductive Step is to get to a form T(n)≤(the expression you are trying to prove)−(another term) so that you can show that T(n)≤(the expression you are trying to prove) when (another term)≥0. (ii) Note that the value of n in any expression is such that n≥n_0.

   Not covered in class. So no need to solve as mentioned in announcement.

9. Solve the following three recurrences using the Master Method and state the order of complexity of the corresponding recursive algorithm. If you simply state the complexity order without showing your work, and the case that applies, you will not get any credit even if the complexity order is correct.

   (a). T(n)=3T(n/2)+n

   Here, a = 3, b = 2

   $\log_b a = \log_2 3 = 1.585 > 1$. So case 1 of master theorem can be applied here.

   $T(n) = \Theta(n^{\log_2 3})$

   (b). $T(n)=3T(n/2)+ n^{\log_2 3}$

   Here, a = 3, b = 2

   $\log_b a = \log_2 3$ which is the power of f(n)

Therefore, by using case 2 of master theorem, T(n) = $\Theta(n^{\log_2 3} \log n)$

(c). T(n)=3T(n/2)+n^3

Here, a = 3, b = 2

$\log_b a = \log_2 3 = 1.585 < 3$

The regularity condition also holds:

$3(\frac{n^3}{8}) \leq k.n^3$ choosing k = 3/8 < 1.

So it follows from the third case of the master theorem.

T(n) = $\Theta(n^3)$