# Assignment 2

January 29, 2022

Questions:

1. The following function is to find the largest value in a given array:

**Algorithm 1** Find Maximum

```
1: procedure FindMax(A)
2:     m → A[0]
3:     while i < n − 1 do                ▷ Search the entire list
4:         if A[i] > m
5:         Then m = A[i]
6:     end while
7:     return m
8: end procedure
```

   1. If value that is bigger than A[0] is found then m is swapped with the new largest value.

   2. In the end if no larger value is found then A[0] is returned.

   3. In the base case n = 1, the function also holds since the loop is never run and just returns m.

   The best case for this algorithm is a for the largest value to be at n=1, but still requires the search of the entire array. The same is true for the worst case, the list will always need to be searched exhaustively. So Big-O, Big-Theta, and Big-Omega are all the same as O(n).

2. Analysis of the recursive function that takes in two numbers and multiples them.

   1. Variable a will always be an even number and always grow larger with each iteration. except on the first iteration where plus a will equal a if b over 2 is odd.

   2. Each iteration b will get smaller until it becomes a multiple of 2n̂ and fall to zero with all evens. Once it becomes a multiple of 2n̂ it continually divide by 2 until it reaches 2, then dividing to 1, and 1 divided by 2 giving zero as it is a floor of the operation. For all odd values of b the value of a is added at the end to get the multiplication of a x b

   3. As an example if a is 30 and b is 5 then a will be $T(1) = 30 \rightarrow T(2) = 60 \rightarrow T(3) = 120 \rightarrow T(4) = 240$ then b will follow the pattern $T(1) = 5 \rightarrow T(2) = 2 \rightarrow T(3) = 1 \rightarrow T(4) = 0$

   4. Once b equals zero is hit the recursive function collapses and is finished. Now each iteration of T(n) where b was odd will be summed and returned to the user. So, for our example T(1) is included since b was equal to 5, T(2) is ignored since b was 2, T(3) is added since b was 1, and T(4) is ignored. Thus T(1) + T(3) = 150, which is indeed the same as a times b.

   The best case scenario is that b is equal to 0 in the first case and completes in one operation. Big-Omega(1). The average case would be Theta(n) since the most common outcome is a set of (a,b) having some linear number of steps before an even number value of b is reached. The upper bond is like O(n) as well since their is always a set number n iterations of T(n) that will give an answer.

3. Solving a recurrence relation using substitution.

$$\begin{array}{ll} T(1) = 1 & n \leqslant 2 \\ T(n) = 2T(n-1) + n - 1 & n > 2 \end{array} \tag{1}$$

   Please see formula sheet on the last page for substitution.

4. Solving using master theorem for recurrence relation:

$$\begin{aligned} T(n) &= c & n \leq 2 \\ T(n) &= 7T\left(\tfrac{n}{2}\right) + n^2 & n > 2 \end{aligned} \tag{2}$$

By master theorem we get that.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \tag{3}$$

By substituting the numbers we get that

$$\log_2 7 > 2 \quad \rightarrow O\left(n^{\log_b a}\right) \tag{4}$$

5. This pseudo code version of merge sort is for merging a list of length n into n lists and arranging them.

**Algorithm 2** Merge Sort

```
1: procedure MergeSort(lst, left, right)
2:     if left ≥ right
3:     return
4:     mid = (left + right)/2
5:     MergeSort(arr, left, mid)
6:     MergeSort(arr, mid + 1, right)
7:     Merge(arr, left, mid, right)
8: end procedure
```

This algorithm will divide the original list into n number of arrays. Then when the list becomes n=1 it will end the recursion and that element will be merged to new list. The recursion will end based on the index where each element should be in the list. The elements of the list are returned and arranged as they are pasted to the merge function.

The time complexity of this algorithm would be

$$O\left(n^{\log_2 3}\right) \tag{5}$$

Since the code follows that a is 2 while b is 2. Thus by master theorem we get this big O.

6. For a more efficient version of polynomial multiplication

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}, \quad 0 \leqslant j \leqslant 2n. \tag{6}$$

if we split all values of A into A zero and one, where zero is the even entries and one is the odd. As well doing the same with B in this manor,

$$A_0(x) = \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor - 1} a_j x^j, \quad A_1(x) = \sum_{j=\lfloor \frac{n}{2} \rfloor}^{n} a_j x^{j - \lfloor \frac{n}{2} \rfloor}. \tag{7}$$

$$A(x)B(x) = \left(A_0(x) + A_1(x)x^{\lfloor \frac{n}{2} \rfloor}\right)\left(B_0(x) + B_1(x)x^{\lfloor \frac{n}{2} \rfloor}\right) \tag{8}$$

This can then be done recursively for every pair of coefficients. Since each

$$a_0 + \left(a_1 x + \left(a_2 x^2 + a_3 x^3\right)\right) ... \tag{9}$$

Giving

$$T(n) = 3T(\frac{n}{2}) + C_n \tag{10}$$

By master theorem is

$$O(n^{\log_2(3)}) \tag{11}$$

$T(1) = 1 \qquad n \leq 2$

$T(n) = 2T(n-1) + n - 1 \qquad n > 2$

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

$= 4T(n-2) + 2n - 4 - n$

$= 4T(n-2) + 3n - 5$

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

$T(n-2) = 2T(n-3) + (n-3)$

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

$= 4T(n-3) + 2n - 6 + n - 1$

$= 4T(n-3) + 3n - 7$

generalize $T(n) = 4T(n-K) + 3n - 3 - 2K$

$K = n + 1$

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

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

$= 4(1) + n - 5$

$T(n) = n - 1$

□