Anand Mahadevan - SID 862132182

CS141 Assignment 5

due Friday, August 13th 11:59 PM

Problem 1.

a)

Call the FindMax function with the Array to return the max element.

```
FindMax2(A):
    n = A.size()
    if (n == 2):
        A.sort()
        return A
    LMax = FindMax2(A[1 : n/2])
    RMax = FindMax2(A[n/2 + 1 : n])
    LRMax = [LMax, RMax]
    LRMax.sort()
    return [LRMax[3], LRMax[4]]

FindMax(A):
    Max2 = FindMax2(A)
    return Max2[2]
```

b)

T(n) = 2T(n/2) + O(1)

Case: $0 < log_2 2 = 1$

T(n) = O($n^{log_2 2}) = O(n^1) = O(n)$

Problem 2.

a)

Case: $1/2 > log_8 2 = 1/3$

T(n) = $O(\sqrt{n})$

b)

Case: $1 < log_3 9 = 2$

T(n) = $O(n^{log_3 9}) = O(n^2)$

c)

Case: $2 < log_2 8 = 3$

T(n) = $O(n^{log_2 8}) = O(n^3)$

d)

Case: $0 < log_2 4 = 2$

T(n) = $O(n^{log_2 4}) = O(n^2)$

Problem 3.
a)
f(n) = n^2
g(n) = $(\sqrt{2})^{lg(n)} = \sqrt{n}$
If we take

$$\lim_{n\to+\infty} f(n)/g(n) = \lim_{n\to+\infty} n^2/\sqrt{n} = \lim_{n\to+\infty} n^{1.5} = \infty$$

This result is significant in that f(n) = $\omega(g(n))$

b)
f(n) = n^n
g(n) = e^n
If we take

$$\lim_{n\to+\infty} f(n)/g(n) = \lim_{n\to+\infty} n^n/e^n = \lim_{n\to+\infty} (n/e)^n = \infty$$

This result is significant in that f(n) = $\omega(g(n))$

c)
f(n) = $n!$
g(n) = 2^n
If we consider how these functions behave as n $\to \infty$, $n!$ adds an additional n term to its product, while 2^n only adds an additional constant 2 term to its product.
So it is very clear that f(n) = $\omega(g(n))$.

d)
f(n) = n^{10}
g(n) = $(lg(n))^{100}$
If we take

$$\lim_{n\to+\infty} f(n)/g(n) = \lim_{n\to+\infty} n^{10}/(lg(n))^{100} =$$

Let's take the derivative of the top and bottom through l'hopital's rule because top and bottom are ∞

$$= \lim_{n\to+\infty} 10n^9/(100 * lg(n)^{99} * 1/(n * ln(2))) =$$

We can take the derivative of top and bottom 99 more times through l'hopital's rule, and removing constant coefficients...

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

This result is significant in that f(n) = $\omega(g(n))$

Problem 4.

T(n) = T(n/2) + 1

$$\begin{array}{c} 1 \\ | \\ 1 \\ | \\ 1 \\ | \\ 1 \end{array}$$

.
.
.

There are lg n of such 1's, as there is a T(n/2) call at every level.

$$T(n) = \sum_{n=0}^{lg(n)} 1 = (lg(n) + 1) * 1 = O(lg(n))$$

Problem 5.

// Strategy: Subdivide the array till 1 or 2 elements, return 0 or A[2] - A[1] respectively
// Split array by two by passing proper indices
// Return of function is largest gain A[j] - A[i] of given Array, low index and high index

```
(MaxGain, low_idx, high_idx) GetMaxGain(A, low_idx, high_idx):
    n = A.size()
    if n == 1:
        return (0,low_idx,high_idx)
    if n == 2:
        return (A[2] - A[1], low_idx, high_idx)

    LMaxGain = GetMaxGain(A, 1, n/2)
    RMaxGain = GetMaxGain(A, n/2+1, n)

    maxGain = max( RMaxGain[1], LMaxGain[1], A[RMaxGain[2]] - A[LMaxGain[3]] ,
        A[RMaxGain[2]] - A[LMaxGain[2]] , A[RMaxGain[3]] - A[LMaxGain[3]] ,
        A[RMaxGain[3]] - A[LMaxGain[2]] )

    if maxGain == (RMaxGain[1]):
        return (maxGain, RMaxGain[2], RMaxGain[3])
    if maxGain == (LMaxGain[1]):
        return (maxGain, LMaxGain[2], LMaxGain[3])
    if maxGain == (A[RMaxGain[2]] - A[LMaxGain[3]]):
        return (maxGain, LMaxGain[3], RMaxGain[2])
    if maxGain == (A[RMaxGain[2]] - A[LMaxGain[2]]):
        return (maxGain, LMaxGain[2], RMaxGain[2])
    if maxGain == (A[RMaxGain[3]] - A[LMaxGain[3]]):
        return (maxGain, LMaxGain[3], RMaxGain[3])
    if maxGain == (A[RMaxGain[3]] - A[LMaxGain[2]]):
        return (maxGain, LMaxGain[2], RMaxGain[3])
```

T(n) = 2T(n/2) + O(n)
Case: 1 = $log_2 2 = 1$
T(n) = $O(n * lg(n))$