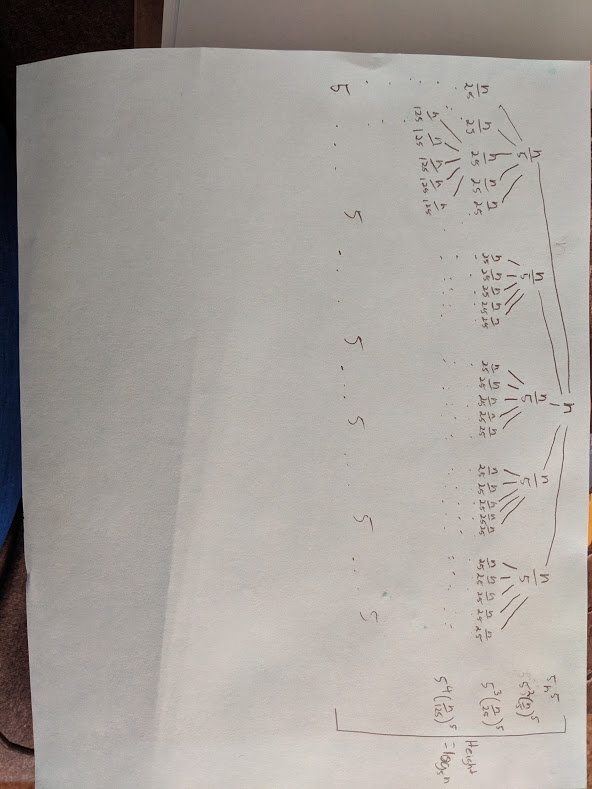
Homework 2

Problem 1

Collaboration: None

Part 1



Height =

Number of leaves =

T(n) = = O()

Part 2

1. T(n) =

Case 3

a = 128, b = 2, f(n) =

n^

f(n) =

T(n) =

1. T(n) =

Case 1

a = 9, b = 3, f(n) =

n^=

f(n) =

T(n) =

1. T(n) =

Cannot Apply master theorem

.

.

.

=

=

T(n) =

1. T(n) =

Case 2

a = 2, b = 2, f(n) =

n^=

f(n) =

T(n) =

Problem 2

Collaboration: None

Part 1

TMS(A[0...n-1])

If n 1, then

Return (A[0...n-1])

Let i = [n/3] and m = [2n/3]

Return Merge(TMS(A[0..k-1]), TMS(A[k...m-1]), TMS(A[m...n-1), A[0...n-1])

Part 2

In the worst case the total number of comparisons would be 5n/3 - 2 O(n)

And the worst case run time would be T(n) = 3T(n/3)+O(n)

Solving this using master theorem would give us T(n) = O()

This tells us that it is not faster than an ordinary Mergesort

Part 3

In order to immediately get a sorted list you would have O(1)

Part 4

The upper bound would be the same as previously and would be T(n) = O()

Part 5

The running time would be better by assumption because you would never have to solve an odd split where you cannot split it by 3. So it would allow it to sort faster.

Part 6

We can assume any form because the theorem does not need to differentiate the power of n.

Problem 3

Collaboration: None

Int getMedian(int a[], int[b], int n){

//base cases

if(n <= 0)

Return -1

if(n == 1)

Return ((a[0] + n[0]) / 2)

if(n == 2)

Return ((max(a[0], b[0]) + min(a[1], b[1])) / 2)

Int m1 = median(a, n)

Int m2 = median (b,n)

if(m1 == m2)

Return m1

if(m1 > m2){

if(n & 1)

Return getMedian(b + (n/2), a, n - (n/2))

Return getMedian(b + (n/2) - 1, a, n - (n/2) + 1)

}

if(m1 < m2){

if(n & 1)

Return getMedian(a + (n/2), b, n - (n/2))

Return getMedian(a + (n/2) - 1, b, n - (n/2) + 1)

}

}

Int median(int a[], int n){

if(n & 1)

Return a[n/2]

Else

Return ((a[n/2] + a[(n/2) - 1])/2)

}

The estimated cost is is O(logn)