Algorithm and Data Structures HW 2

Mario Alberto Hernandez Salamanca

February 22, 2019

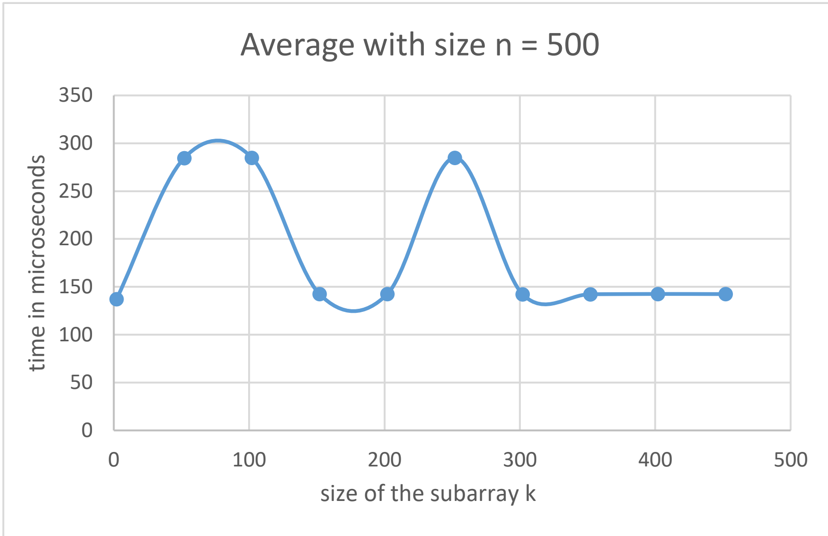
1 Merge Sort

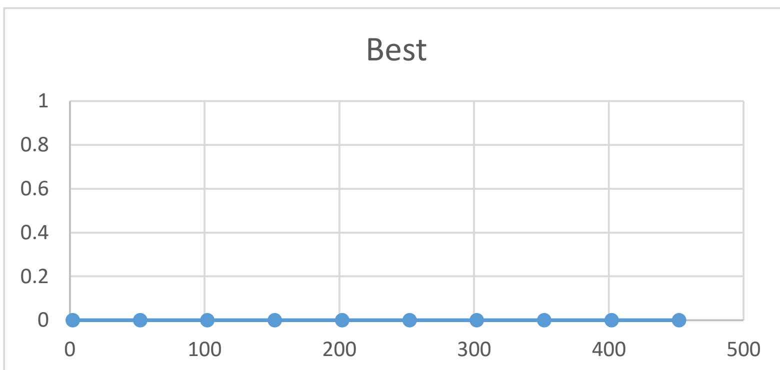
1.1 Merge Sort Variant

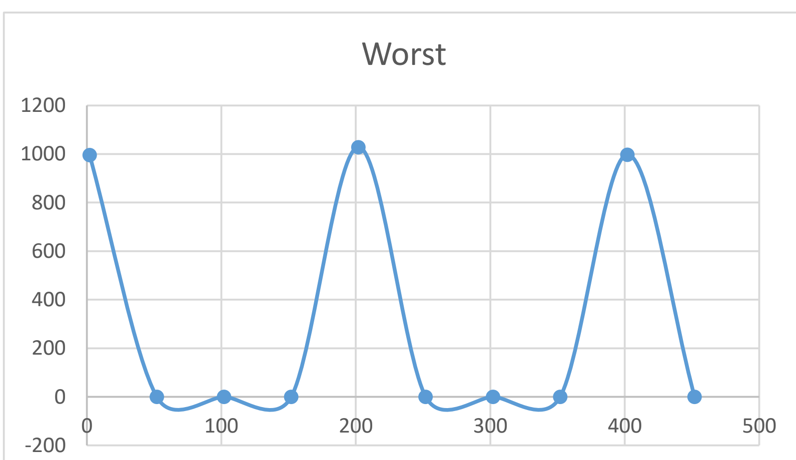
```
void mergeSort(int arr[],int sub ,int l, int r)
{
        //sub is the size k of the sub array
        //in witch we will applied insertion sort
    int size = r - l + 1 ;

    if(sub <= size){
        insertion_sort(arr, l , r);
    }
    else
    {

        int m = l+(r-l)/2;

        mergeSort(arr, sub ,l, m);
        mergeSort(arr, sub ,m+1, r);

        merge(arr, l, m, r);
    }
}
```

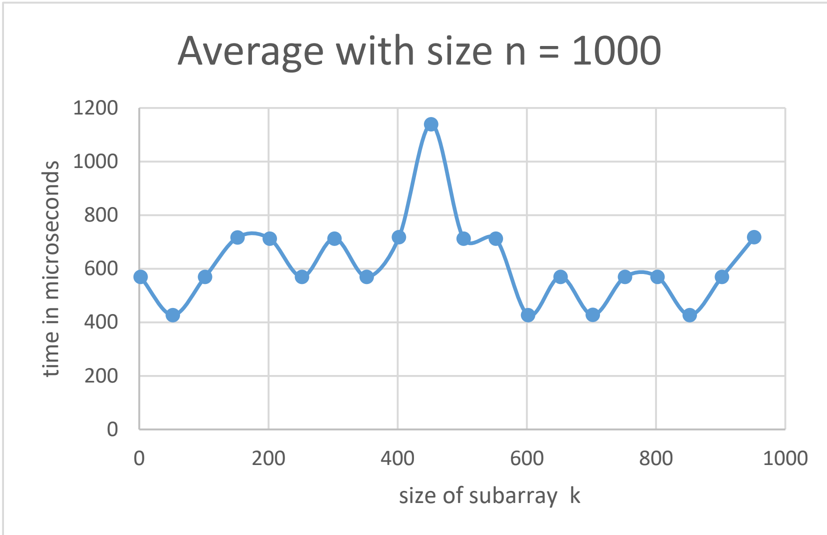
1.2 Plot of running time of the algorithm

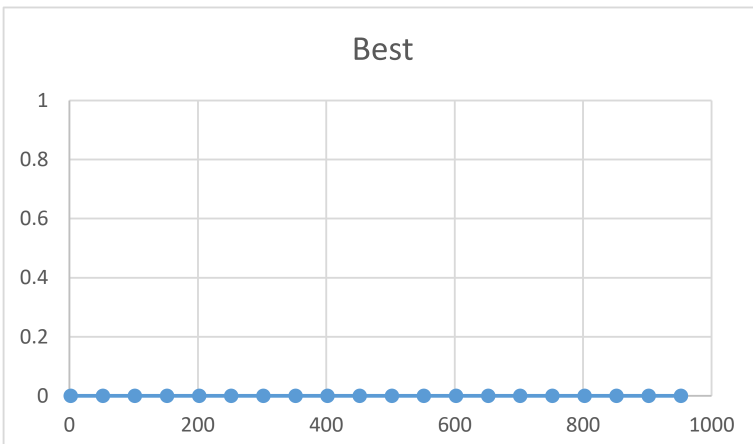
I am using 3 different graphs for each case, each graph represent the sorting algorithm applied on an array that have a size constant $500 < n < 5000$ with $2 < k < n$

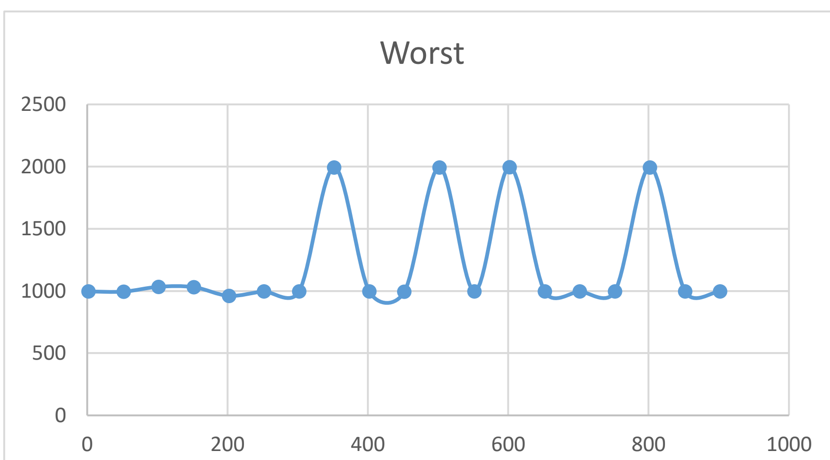

Average with size n = 500
350
300
250
200
150
100
50
0
time in microseconds
0
100
200
300
400
500
size of the subarray k

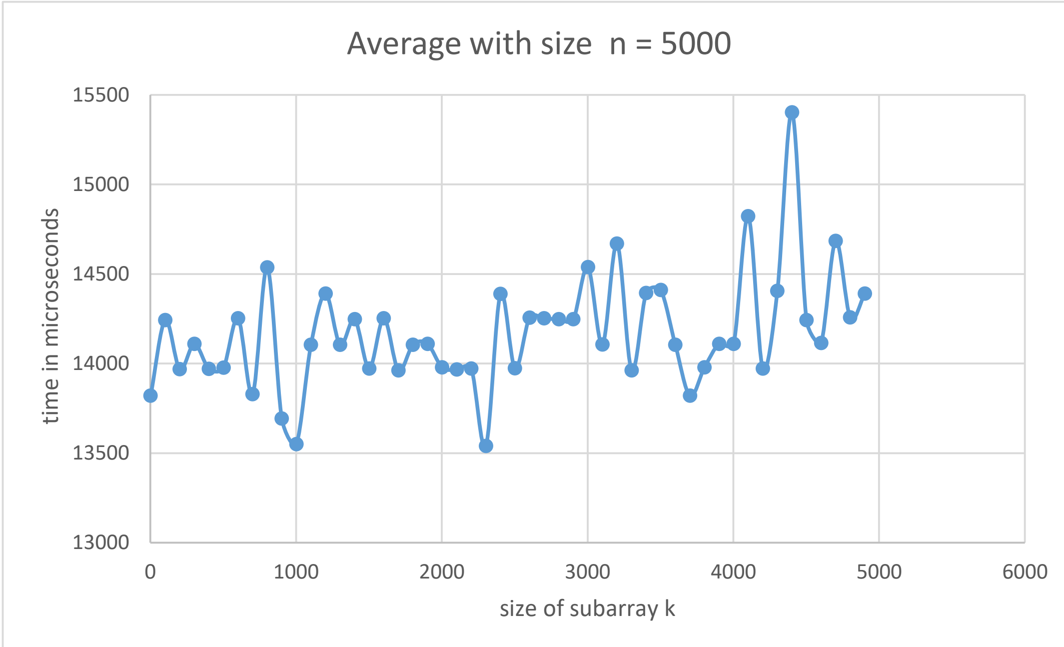

Best
1
0.8
0.6
0.4
0.2
0
0
100
200
300
400
500

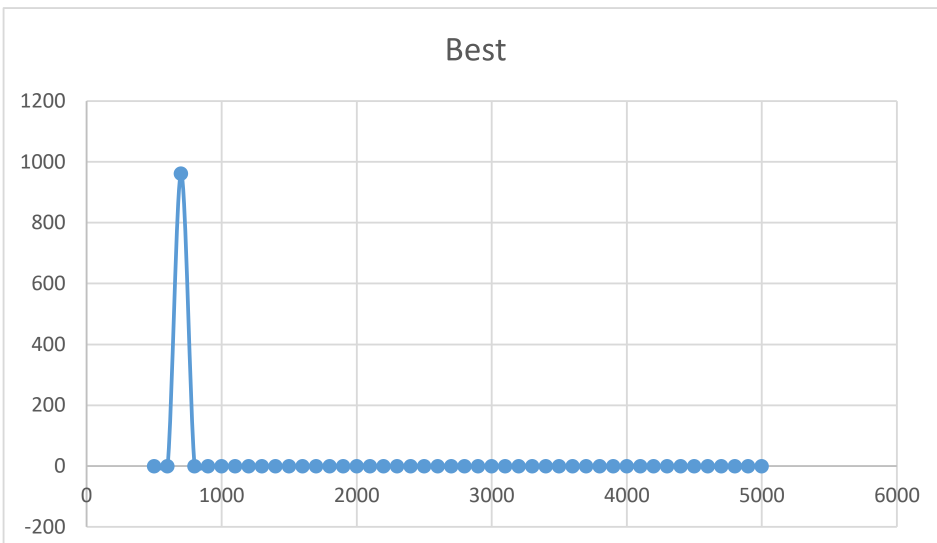

Worst
1200
1000
800
600
400
200
0
-200
0
100
200
300
400
500

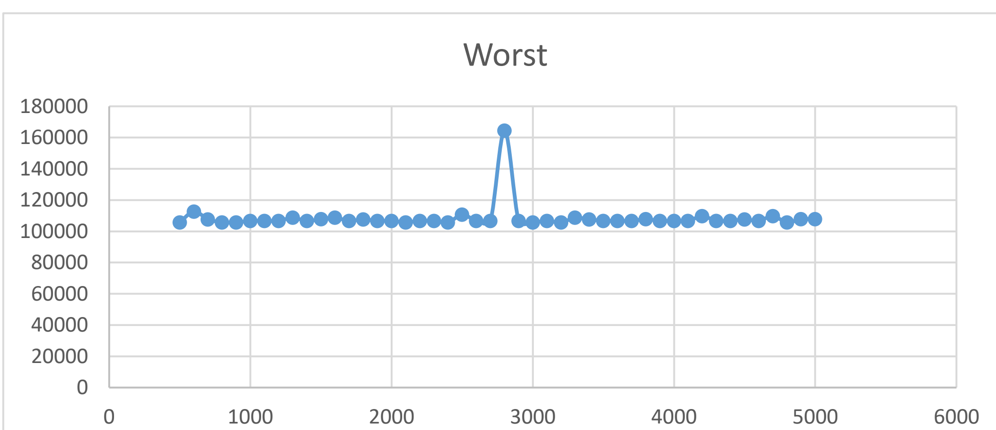

Average with size n = 1000
time in microseconds
size of subarray k
0
200
400
600
800
1000
1200
0
200
400
600
800
1000


Best
0
0.2
0.4
0.6
0.8
1
0
200
400
600
800
1000


Worst
0
500
1000
1500
2000
2500
0
200
400
600
800
1000


Average with size n = 5000
time in microseconds
15500
15000
14500
14000
13500
13000
0
1000
2000
3000
4000
5000
6000
size of subarray k


Best
1200
1000
800
600
400
200
0
-200
0
1000
2000
3000
4000
5000
6000


Worst
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1000
2000
3000
4000
5000
6000

1.3 Graph analysis

As we can see on the graphs of the average cases in most of them when k increases is not a predictable tendency of the running time but when k is bigger the algorithm works more as an insertion sort algorithm instead of mergesort but when k is small is more like a mergesort.

For the Best case the algorithm is really lineal because as we know insertion sort in the best case it has a lineal order but in some cases when n is really big and k is between $n/2$ sometimes the time has variations so k does not affect anything at all.

Worst Case is the most interesting the peaks of the worst case graph is when k gets higher the algorithm works as a insertion sort being $O(n^2)$ compare to the mergesort $O(nlgn)$ is slightly slower in those parts.

1.4 How to chose k

In practice from my point of view the algorithm k should have a value between 2 and 30 because is when the insertion sort is pretty fast and makes the merge way simpler so you do not have the necessity of doing to many swaps

2 Recurrences

2.1 A

$$T(n) = 36T(n/6) + 2n$$

$a = 36$ $b = 6$
$n^{log_6 36} = n^2$
$F(n) \in O(n^{2-\epsilon})$ for $\epsilon = 1$

$$\text{thus } T(n) = \Theta(n^2)$$

2.2 B

$$T(n) = 5T(n/3) + 17n^{1.2}$$

$a = 5$ $b = 3$
$n^{log_3 5}$, $log_3 5 > 1.2$
and we have that $log_3 5 - 1.2 > 0$
$F(n) \in O(n^{log_3 5-\epsilon})$ for $\epsilon = (log_3 5 - 1.2)$

$$\text{thus } T(n) = \Theta(n^{log_3 5})$$

2.3 C

$$T(n) = 12T(n/2) + n^2 lg(n)$$

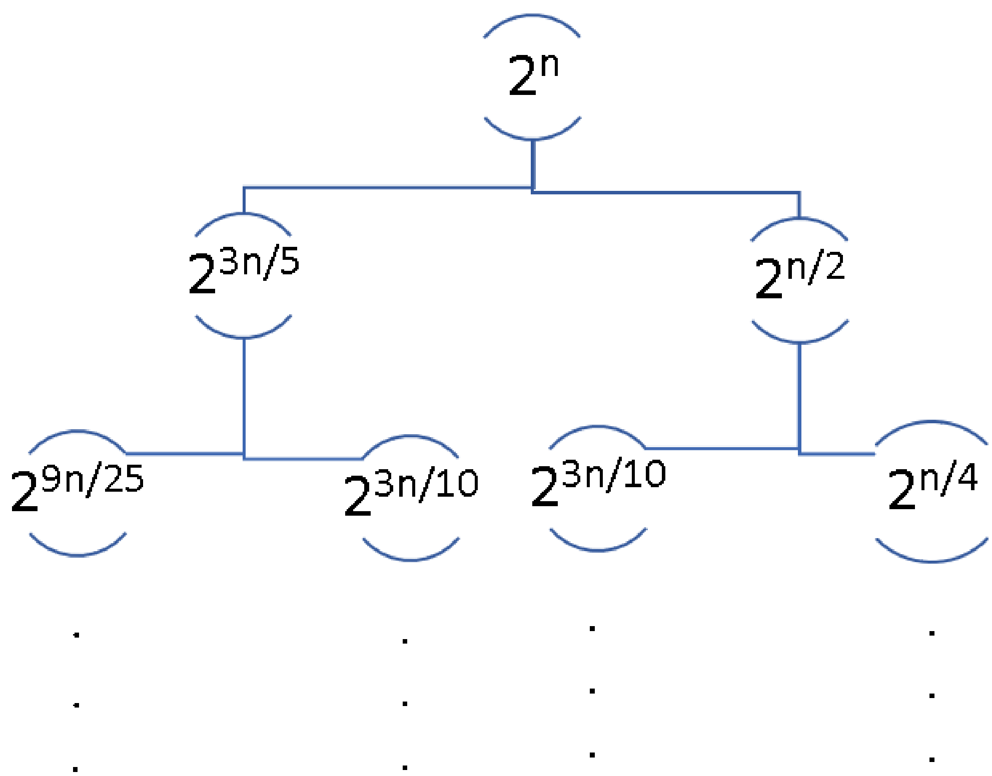
$a = 12$ $b = 2$
$n^{log_2 12}$, $n > log_2 n$
$F(n) \in O(n^{log_2 12-\epsilon})$ for $\epsilon = 1$

$$\text{thus } T(n) = \Theta(n^{log_2 12})$$

2.4 D

$$T(n) = 3T(n/5) + T(n/2) + 2^n$$

for this problem I am using recursion tree



for the first 3 levels we got the following sums

$$2^n + 2^{3n/5} + 2^{9n/25} + 2^{3n/10} + 2^{3n/10} + +2^{n/4} + ...$$

we take common factor n^2 we get the following

$$2^n(1 + 2^{(-2n/5)} + 2^{(-16n/25)} + 2^{(-7n/10)} + 2^{(-7n/10)} + 2^{(-3n/4)} + ...)$$

we are interested when n goes to infinity so we get the following

$$2^n * (1 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + ...)$$

$$2^n$$

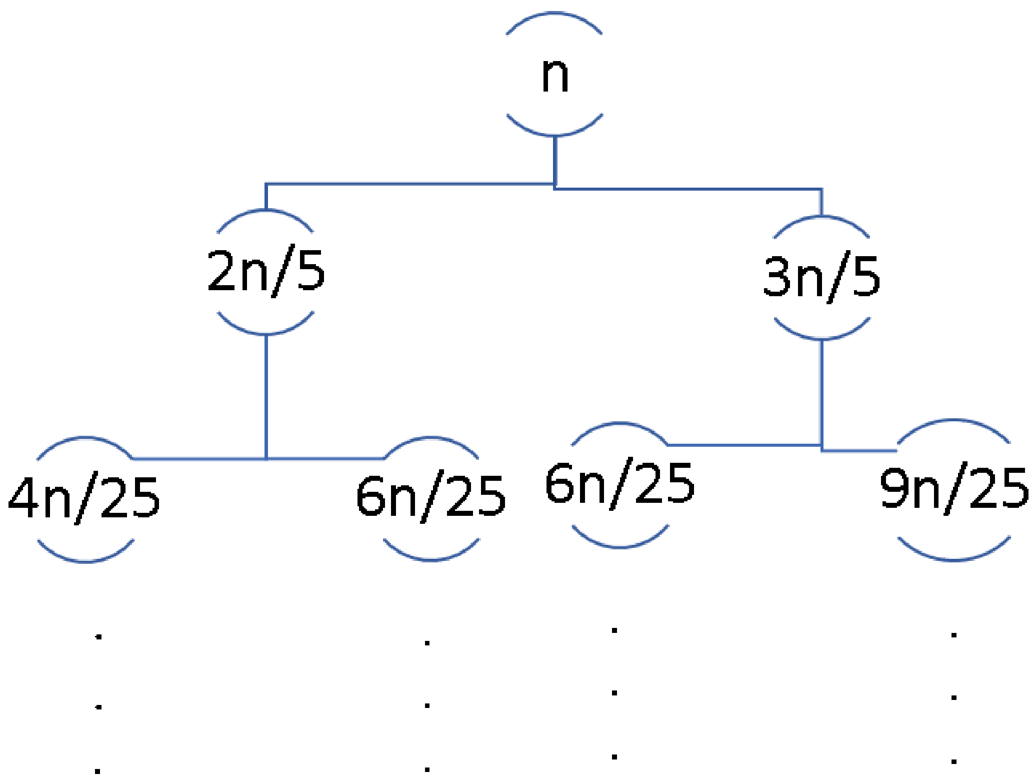
thus

$$T(n) \in \Theta(2^n)$$

2.5 E

$$T(n) = T(2n/5) + T(3n/5) + \Theta(n)$$

we can assume that $\Theta(n)$ is a representation of how the $f(n)$ behaves so I take $f(n) = cn$ being c a constant but at the end it does not affect the overall order so I will use $f(n) = n$ so I will start my recursion three using n as my root



for the first 3 levels we got the following sums

$$n + n + n + ...$$

as the tree continues growing we see that is a sum of n and as we can know the high of the tree because it divide in 5 branches every time the high is $log_5 n$ thus

$$T(n) \in \Theta(nlogn)$$