CS301 HW1

Adnan Kaan Ekiz - 20848

February 2019

1 Problem 1

Let's assume that:
$g_1 = n2^n$, $g_2 = nlogn$, $g_3 = n!$, $g_4 = n$, $g_5 = n^{100}$
$g_6 = 2^n$, $g_7 = log(n)$, $g_8 = log(n!)$, $g_9 = (log(n))!$, $g_{10} = 2^{2^n}$

Order between these notation's should be:

- $log(n) = O(n) \rightarrow g_7 = O(g_4)$
- $n = O(log(n!)) \rightarrow g_4 = O(g_8)$
- $log(n!) = O(nlogn) \rightarrow g_8 = O(g_2)$
- $nlogn = O((logn)!) \rightarrow g_2 = O(g_9)$
- $(logn)! = O(2^n) \rightarrow g_9 = O(g_6)$
- $2^n = O(n!) \rightarrow g_6 = O(g_3)$
- $n! = O(n2^n) \rightarrow g_3 = O(g_1)$
- $n2^n = O(2^{2^n}) \rightarrow g_1 = O(g_{10})$
- $2^{2^n} = O(n^{100}) \rightarrow g_{10} = O(g_5)$

2 Problem 2

2.1 $T(n) = 2T(n/2) + n^3$

Guess that the solution is $T(n) = \Theta(n^3)$,
Our method is to prove that $T(n) \leq cn^3$ for an appropriate choice of $c > 0$.
By using substitution method:

$T(n) = 2T(n/2) + n^3$
$= 2c(n/2)^3 + n^3$
$= 1/4cn^3 + n^3$

$= cn^3 - (3/4cn^3 - n^3)$
$\leq cn^3$
so the last step holds for $c \geq 4/3 \, and \, n > 0$

2.2 $T(n) = 7T(n/2) + n^2$

Guess that the solution is T(n) = $\Theta(n^{log_2 7})$,
Our method is to prove that T(n) $\leq c_1 n^{log_2 7} - c_2 n^2$ for an appropriate choice of $c_1 > 0$ and $c_2 > 0$.
By using substitution method:

$T(n) = 7T(n/2) + n^2$
$\leq 7[c_1(n/2)^{log_2 7} - c_2(n/2)^2] + n^2$
$= c_1 n^{log_2 7} - c_2 n^2 - (3/4c_2 n^2 - n^2)$
$\leq c_1 n^{log_2 7} - c_2 n^2$
so the last step holds for $(3/4c_2 - 1)n^2 \geq 0$. Therefore $c_2 \geq 4/3$ and $n > 0$.

2.3 $T(n) = 2T(n/4) + \sqrt{n}$

Guess that the solution is T(n) = $\Theta(n^{1/2} logn)$,
Our method is to prove that T(n) $\leq cn^{1/2} logn$ for an appropriate choice of $c > 0$.
By using substitution method:

$T(n) = 2T(n/4) + n^{1/2}$
$\leq 2c(n/4)^{1/2} log(n/4) + n^{1/2}$
$= cn^{1/2} logn - (cn^{1/2} log4 - n^{1/2})$
$\leq cn^{1/2} logn$
so the last step holds for $(clog4 - 1)n^{1/2} \geq 0$. Therefore $c \geq 1/log4$ and $n > 0$.

2.4 $T(n) = T(n-1) + n$

In this case, I apply recursion tree method to solve the recurrence problem:
$T(n) = T(n-1) + n$
$= T(n-2) + n - 1 + n$
$= T(n-3) + n - 2 + n - 1 + n$
....
$= T(0) + 1 + 2 + ... + n - 1 + n = \sum_{i=1}^{n} i = n(n+1)/2 = O(n^2)$

3 Problem 3

3.1 a)

3.1.1 (i)

O(logn)

3.1.2 (ii)

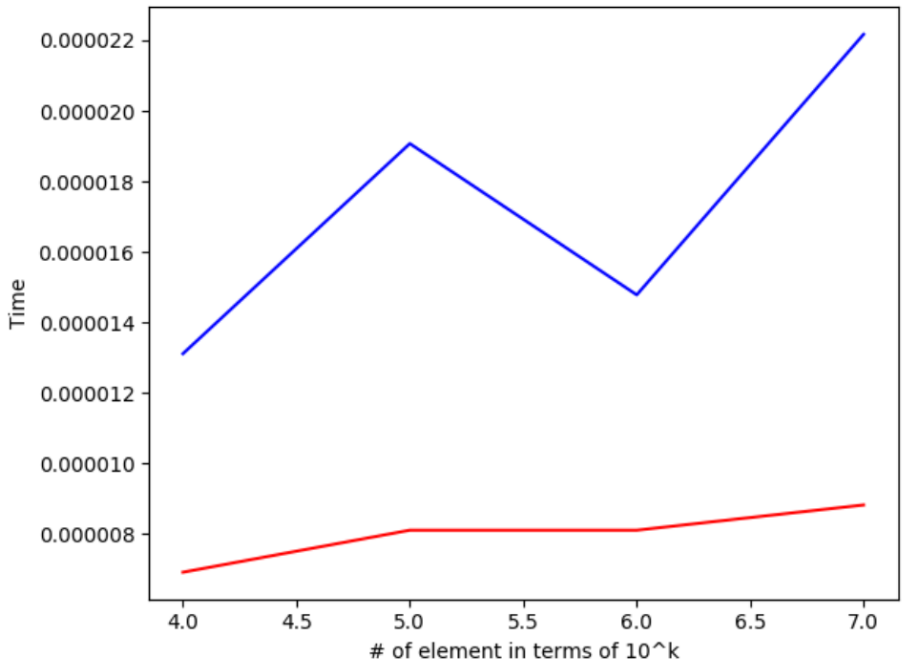
T(n) = T(n/2) + 1
$= T(n/4) + 1 + 1$
$= T(n/2^k) + 1 + 1 + ... + 1$
After k steps our search space will be exhausted. Then,
$n/2^k = 1$
$n = 2^k \rightarrow k = log_2 n$
Therefore time complexity of binary search algorithm is $O(log_2 n)$.

3.2 b)

3.2.1 (i)

Algorithm	10^4	10^5	10^6	10^7
Iterative	6.914e-06	6.914e-06	7.152e-06	8.106e-06
Recursive	1.573e-05	1.621e-05	1.478e-05	2.217e-05

3.2.2 (ii)



Blue is for the time analysis of recursive binary search algorithm.
Red is for the time analysis of iterative binary search algorithm.

3.2.3 (iii)

Algorithm's running complexity can change based on the data type given in the experiment. Therefore, I do believe that it is not really efficient to scale the algorithms based on the running time. We can also give an assumption about the data size being small, hence it will be scalable under the given assumption.

3.2.4 (iv)

Result that I have get confirms the theoretical results that I have found in (a). As the number of element increases, time of the algorithm also increases at the same time. There might be some small disturbance in the graph for recursive algorithm but it is still consistent with what I have found in general. Both of the functions have logarithmic patterns.

3.3 c)

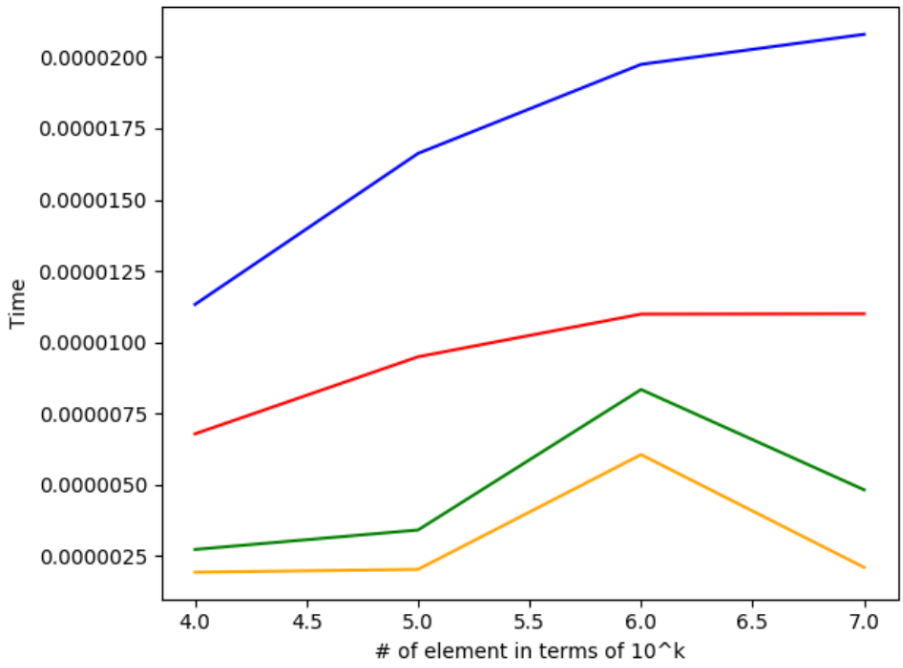
3.3.1 (i)

Algorithm	10^4	10^5	10^6	10^7
Iterative				
Mean(μ)	6.785e-06	9.489e-06	1.098e-05	1.099e-05
Std(σ)	1.932e-06	2.032e-06	6.057e-06	2.102e-06
Recursive				
Mean(μ)	1.132e-05	1.662e-05	1.974e-05	2.079e-05
Std(σ)	2.731e-06	3.413e-06	8.340e-06	4.822e-06

CPU : 3.1 ghz
RAM : 8 GB
OS : macOS mojave

3.3.2 (ii)

Blue line is for the mean of recursive function.
Red line is for the mean of iterative function.
Green line is for the standard deviation of recursive function.
Orange line is for the standard deviation of iterative function.



3.3.3 (iii)

It is still consistent with the worst case data in part(b). Iterative binary search algorithm has still smaller mean time compared to the one with the recursive binary search algorithm's mean. Since its taking a random key to search for, average time is slightly smaller than the worst case results in general.

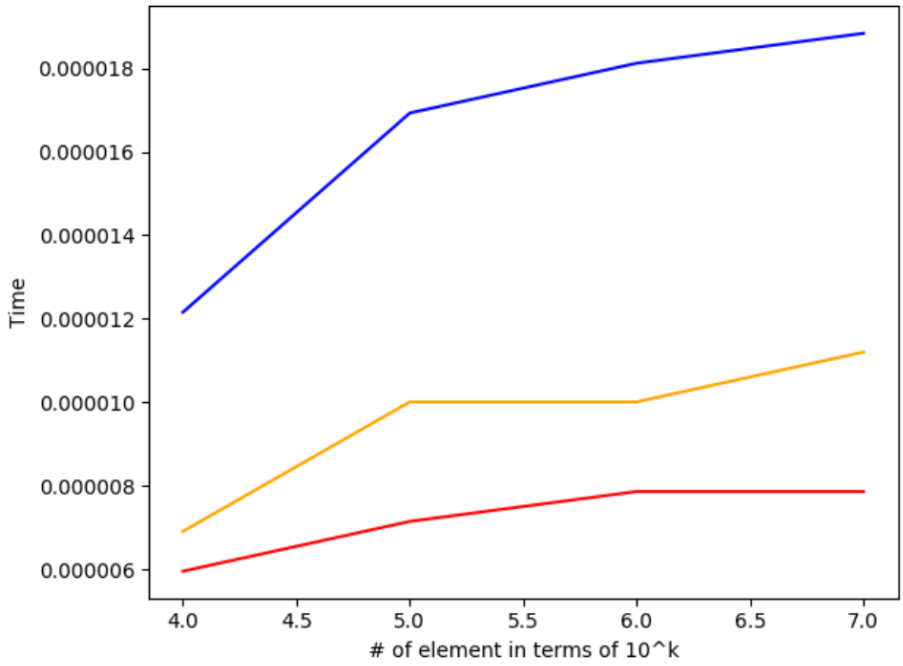
3.4 d)

We can improve the running time of the recursive binary search algorithm by removing the list slice actions since each time slice is used it creates a new array, hence decreasing the running time dramatically.

On the other hand improved version of the recursive binary search algorithm does not create any new list and does the processes on the list that we already created.

```python
def improved_r_binarySearch(alist, l, r, item):
    if r >= l:
        midpoint = l + ((r - l)//2)
        if alist[midpoint] == item:
            return True
        elif alist[midpoint] > item:
            return improved_r_binarySearch(alist, l, midpoint-1, item)
        else:
            return improved_r_binarySearch(alist, midpoint+1, r, item)
    else:
        return False
```

Blue line is for the initial recursive binary search algorithm.
Orange line is for the improved recursive binary search algorithm.
Red line is for the iterative binary search algorithm.



It can be seen in the graph that by improving the code a little bit, recursive binary search algorithm can be improved in terms of time.