CS 301 – Algorithms

Homework 1 – 04/03/2019

Atilla Alpay Nalcaci - 19510

Problem 1. Asymptotic Growth

Arrangement of the functions $n2^n$, $n\log(n)$, $n!$, n, n^{100}, 2^n, $\log(n)$, $\log(n!)$, $(\log(n))!$, $2^{2^\wedge n}$ are provided below;

$$2^{2^\wedge n} > n! > n^{100} > n2^n > 2^n > (\log(n))! > n\log(n) > \log(n!) > n > \log(n)$$

Problem 2. Recurrences

For this question, consider the below notations in order to compare and contrast the applied theorem and the questions;

- Master's Theorem

  $T(n) = aT(n/b) + f(n)$ where $a \geq 1$, $b > 1$ and $f(n) = \theta(n^{\log_b a})$

  Case 1: $\log_b a > k$ then $O(n^{\log_b a})$

  Case 2: $\log_b a = k$

  ↳ If $p > -1$, $\theta(n^k \log^{p+1} n)$

  ↳ If $p = -1$, $\theta(n^k \log\log n)$

  ↳ If $p < -1$, $\theta(n^k)$

  Case 3: $\log_b a < k$

  ↳ If $p \geq 0$, $\theta(n^k \log^p n)$

  ↳ If $p < 0$, $O(n^k)$

Assuming $T(n)$ is constant for $n \leq 2$.

(a) $T(n) = 2 \,.\, T(n / 2) + n^3$

- Using the Master's Theorem; a = 2, b = 2 and $f(n) = \theta(n^k \log^p n)$ since $\log_b a < k$ ($\log_2 2 < 3$) where k = 3 and p = 0 ($f(n) = n^3$).

  ↳ $p = 0 : \theta(n^3) \rightarrow T(n) = \theta(n^3)$

(b) $T(n) = 7 \,.\, T(n / 2) + n^2$

- Using the Master's Theorem; a = 7, b = 2 and $f(n) = O(n^{\log_b a})$ since $\log_b a > k$ ($\log_2 7 > 3$) where k = 2 and p = 0 ($f(n) = n^2$).

  ↳ $p = 0 : \log_2 7 > 2 \rightarrow T(n) = O(n^{\log_2 7})$

(c) $T(n) = 2 \,.\, T(n/4) + \sqrt{n}$

- Using the Master's Theorem; a = 2, b = 4 and $f(n) = \theta(n^k \log^{p+1} n)$ since $\log_b a = k$ ($\log_4 2 = 0.5$) and $p > -1$ ($0 > -1$) where k = 0.5 and p = 0 ($f(n) = \sqrt{n}$).

  ↳ $p > -1 : \theta(n^{0.5} \log n)$

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

- Master's Theorem is not suitable for this recurrence. Using Substitution Method;

  ↳ *Guess* $T(n) = O(n)$

  ↳ *Now Assume "*$T(k) \leq c.k$*" and prove "*$T(n) \leq c.n$*"*

  ↳ $T(n) = \underbrace{T(n-1) + n}$

  $\leq \underbrace{c.n - 1 + n}$

  $= \underbrace{c.n}_{\text{Desired}} - \underbrace{(c - n)}_{\text{Residual}} \rightarrow$ $(c - n) \geq 0$ if $c \geq n$ *and thus proving* $T(n) = O(n)$

<u>Problem 3.</u> Binary Search – Python

(a)

Iterative Binary Search

```
def binarySearch(alist,item):
     first = 0
     last = len(alist)-1
     found = False

     while first<=last and not found:
       midpoint = (first + last)/2
       if alist[midpoint] == item:
         found = True
       else:
         if item < alist[midpoint]:
            last = midpoint-1
         else:
            first = midpoint+1
     return found
```

Recursive Binary Search

```
def binarySearch(alist,item):
    if len(alist) ==  0:
       return False
    else:
       midpoint = len(alist)/2
       if alist[midpoint] == item:
         return True
       else:
         if item<alist[midpoint]:
            return binarySearch(alist[:midpoint],item)
         else:
            return binarySearch(alist[midpoint+1:],item)
```

(i) For the Iterative Binary Search, size of the subarray splits (e.g. assuming the size of the array is 2^n, at each iteration, the size will shrink as 2^{n-1}, 2^{n-2}, …, 2^{n-n}) at each iteration. Iterations terminate when the subarray length reaches 2^{n-n}. Thus, for a given array with the size of n, $\log(n) + 1$ operations occur, resulting in $O(\log(n))$.

(ii) For the recurrence;

$$T(n) = \begin{cases} 1 & , \quad n = 1 \\ T(n / 2) + O(n) + 1 & , \quad n > 1 \end{cases}$$

The asymptotic running time of the recursive binary algorithm is as follows:

- Using Substitution Method;

  ↳ $T(n) = T(n / 2) + n + 1 = T(n / 4) + 3n / 2 + 2 = T(n / 8) + 7n / 4 + 3$

  $$= \ldots = T(n / 2^k) + \frac{(2^k - 1) \,.\, n}{2^{k-1}} + k$$

  ↳ $T(n / 2^k) = 1 \rightarrow n = 2^k$ where $k = \log(n)$

  ↳ By plugging in $k = \log(n)$ we get $T(n) = \underbrace{1 + 2n - 2 + \log(n)}_{O(n + \log(n)) = O(\log(n))}$

(b)

(i) Computer Specs:

CPU – Intel® Core™ i7-6700HQ (6M Cache, up to 3.50 GHz)

Chipset – Intel® HM170

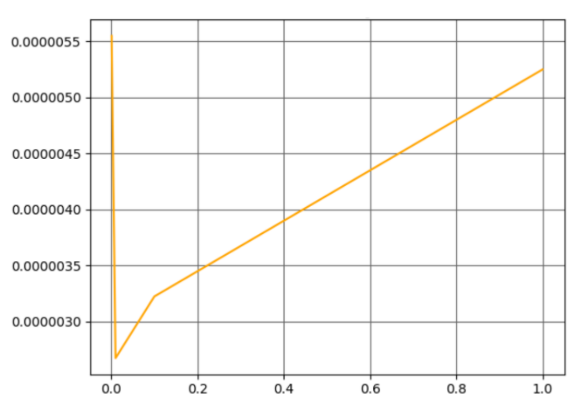
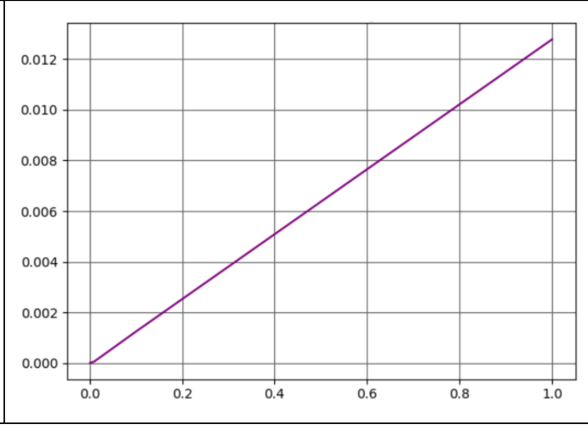
Memory – 32 GB DDR4

Storage – 512 GB SSD + 1 TB SATA HDD

Operating System – Windows 10 Home

Algorithm	$n = 10^4$	$n = 10^5$	$n = 10^6$	$n = 10^7$
Iterative	2.2200000000083264e-05	1.0700000000252174e-05	1.2900000000204415e-05	2.1000000000270802e-05
Recursive	4.399999999993298e-05	0.00026230000000015963	0.0049815999999998080	0.05110020000000004

(ii) Graphics for experimental results are provided below;

Iterative Binary Search	Recursive Binary Search
	

x-axis: Input Size, y-axis: Running Time (in seconds)

(iii) Considering the scalability with respect to these experimental results, the given binary search algorithms are sufficient for large numbers. The given algorithms sustain O(n) with high efficiency.

(iv) Considering the theoretical results from part (a), the experimental results are unable to confirm due to the fact that resulting complexity was O($n$log(n)) whereas it is found to be O(n) for experimental results.

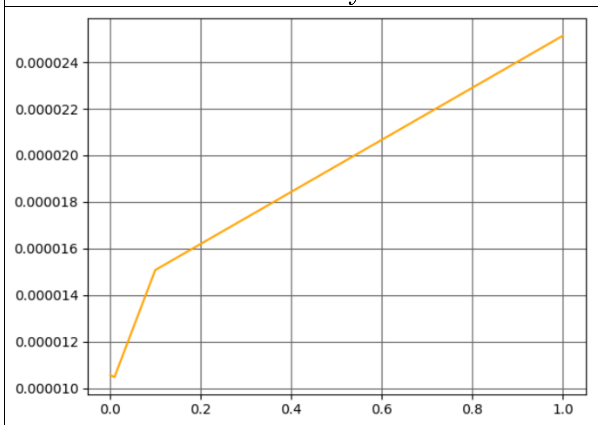
(c)

(i) Average running times in seconds (μ) and the standart deviation (σ) of the algorithms are provided below:

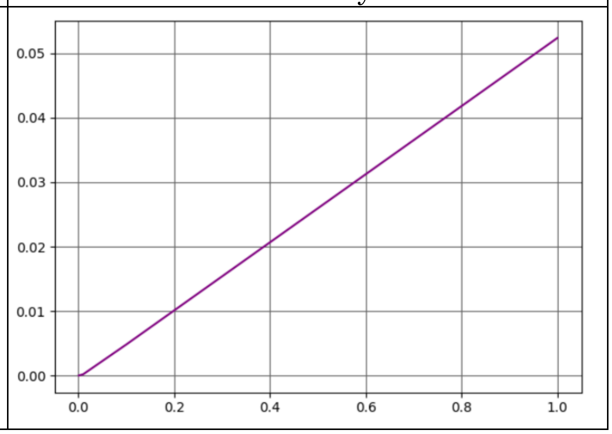
Algorithm	$n = 10^4$		$n = 10^5$		$n = 10^6$		$n = 10^7$	
	μ	σ	μ	σ	μ	σ	μ	σ
Iterative	1.0563999999999574e-05	0.0001591251931849304	1.0480000000000826e-05	0.000152100493026132	1.5074000000012689e-05	0.00018693260243311192	2.5134000000059943e-05	0.0002917690792085757
Recursive	3.702800000000561e-05	0.0005527849152378313	0.00022527200000000747	0.003353920956317864	0.004839922000000008	0.06914589129671335	0.052353583999999974	0.6803002816117664

(ii) Graphics for experimental results are provided below;

Iterative Binary Search



Recursive Binary Search



x-axis: Input Size, y-axis: Running Time (in seconds)

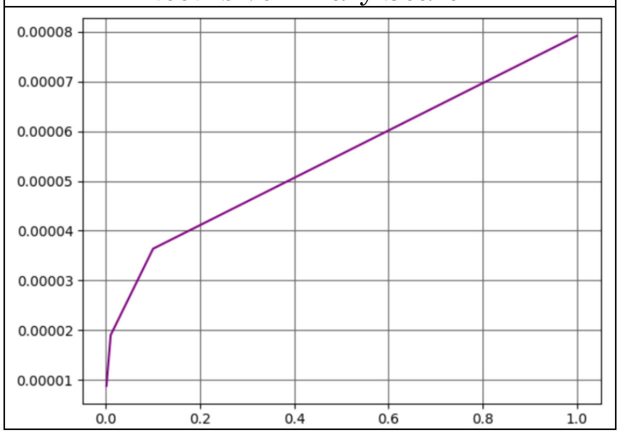
(iii) Considering the average running times, Recursive Binary Search has almost no difference whereas Iterative Binary Search has a drastic increase of rate but not in values.

(d)

In order to improve Recursive Binary Search algorithm so that it can have the same asymptotic running time as the Iterative Binary Search algorithm, `start` and `end` points of the indexes can be represented as a parameter in order to avoid using `slice` function.

- Since aim is to improve the recursion, it can be observed experimentally below:

Recursive Binary Search



x-axis: Input Size, y-axis: Running Time (in seconds)

Ultimately, the modified recursive binary search algorithm serves as expected as it can be observed that the new experimental results are far more familiar with the iterative binary search algorithm than the prior experiment.