Cavit Çokur
HW 1
23657

Problem 1

logn = O(n)

n = O(log(n!))

log(n!) = O(nlogn)

nlogn = O(n^100)

n^100 = O((log(n))!)

(log(n))! = O(2^n)

2^n = O(n2^n)

n2^n = O(n!)

n! = O(2^(2^n))

Problem 2

a) $a=2$, $b=2$ $\Rightarrow n^{\log_b a} = n$, $f(n) = n^3$

↳ Case 3 = $f(n) = \Omega(n^{\varepsilon}) \longrightarrow \varepsilon = 3$

and $2(n/2)^3 \leq cn^3$ for $c = 1/4$

↳ $T(n) = \Theta(n^3)$

b) $a=7$, $b=2$ $\Rightarrow n^{\log_2 7}$, $f(n) = n^2$

$n^{\log_2 4 + \log_2 3}$ → Bigger than $f(n)$

↳ Case 1 $\Rightarrow f(n) = O(n^{2+\log_2 3 - \varepsilon}) \longrightarrow \varepsilon = \log_2 3$

↳ $T(n) = \Theta(n^{\log_2 7})$

c) $a = 2$, $b = 4$ $\Rightarrow n^{\log_4 2}$, $f(n) = n^{1/2}$ C

Same order

$\underline{\text{Case 2}} \Rightarrow f(n) = \Theta(n^{\log_4 2} \log^{k+1} n) \rightarrow k = 0$

$\hookrightarrow T(n) = \Theta(n^{\log_4 2} \log n)$

d)

$$\left.\begin{array}{c} n \\ \downarrow \\ n-1 \\ \downarrow \\ n-3 \\ \vdots \\ \Theta(1) \end{array}\right\} \text{Total} = \Theta(n^2)$$

guess $\Rightarrow O(n^2)$

assume $\Rightarrow T(k) \leq ck^2$ for $k < n$

prove $\Rightarrow T(n) \leq cn^2$ by induction

$\hookrightarrow T(n) = T(n-1) + n$

$\leq c(n-1)^2 + n = c.(n^2 - 2n + 1) + n$

$= cn^2 - (c(2n-1) - n)$ ← desired − residual

$\leq cn^2$ whenever $(2n-1).c - n \geq 0$

$2cn - c - n \geq 0$

$n(2c-1) - c \geq 0$

$$\boxed{\begin{array}{c} c \geq 1 \\ n \geq 1 \end{array}}$$

$\hookrightarrow T(n) = O(n^2)$

Problem 3

a)

(i)List being divide to half in every iteration because of "midpoint = (first + last) // 2 " this line so the upper bound will be O(logn). At worst-case which is the case that the item is not in the list, running time will be O(logn).

(ii)It is T(n/2) + O(n)

```
if item<alist[midpoint]:
  return binarySearch(alist[:midpoint],item)
else:
   return binarySearch(alist[midpoint+1:],item)
```

this if, else gives the T(n/2) and alist[:midpoint] this copy operation gives o(n)

I used master theorem: a = 1 , b = 2 n^log(1 base 2) will be n to 0 which is 1. We got the case 3 because f(n) dominates. Running time will be O(n).

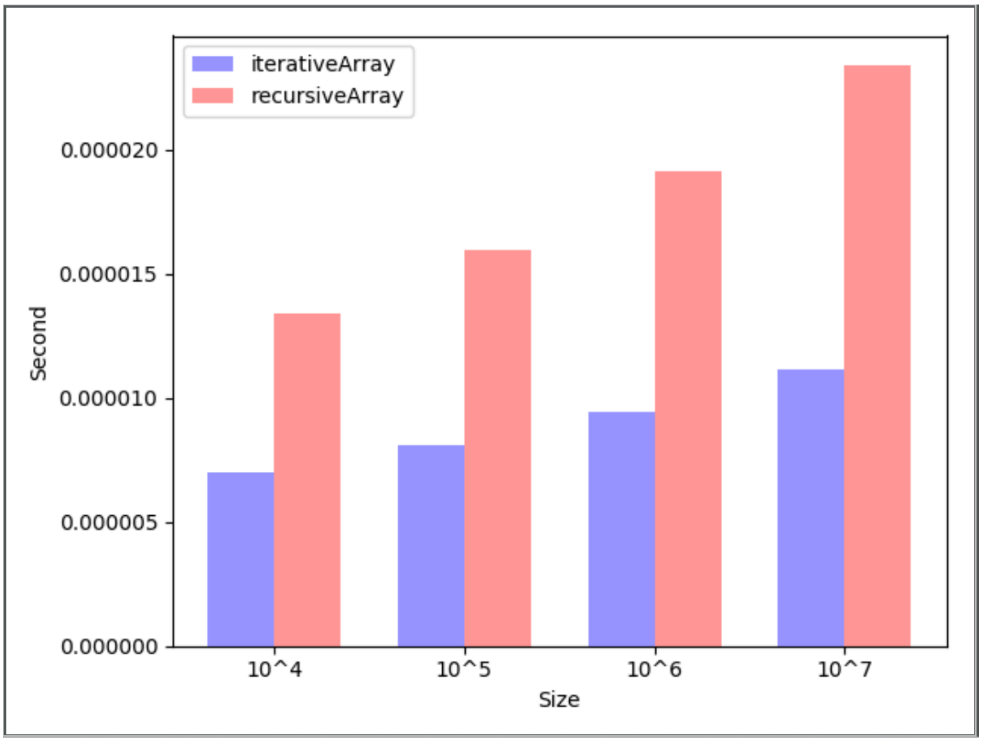
b)(i)

Algorithm	$n = 10^4$	$n = 10^5$	$n = 10^6$	$n = 10^7$
Iterative	6.8 microsec	8.1 microsec	10 microsec	11.3 microsec
Recursive	13 microsec	16.4 microsec	20.4 microsec	24.7 microsec

Properties:

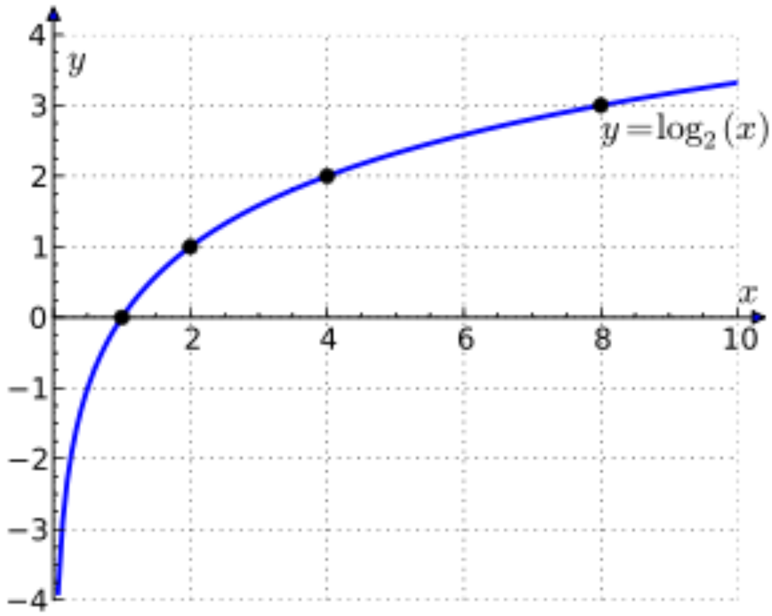
- MacBook Pro (Early 2015)
- MacOS
- 2.7 GHz Intel Core i5
- 8 Gb 1867 MHz DDR3
- Intel Iris Graphics 6100 1536 MB
- Pycharm

(ii)



(iii)
Complexity of bad recursive function is o(n) and iterative's is O(logn) but in practice they are slightly different as we can see in graph.

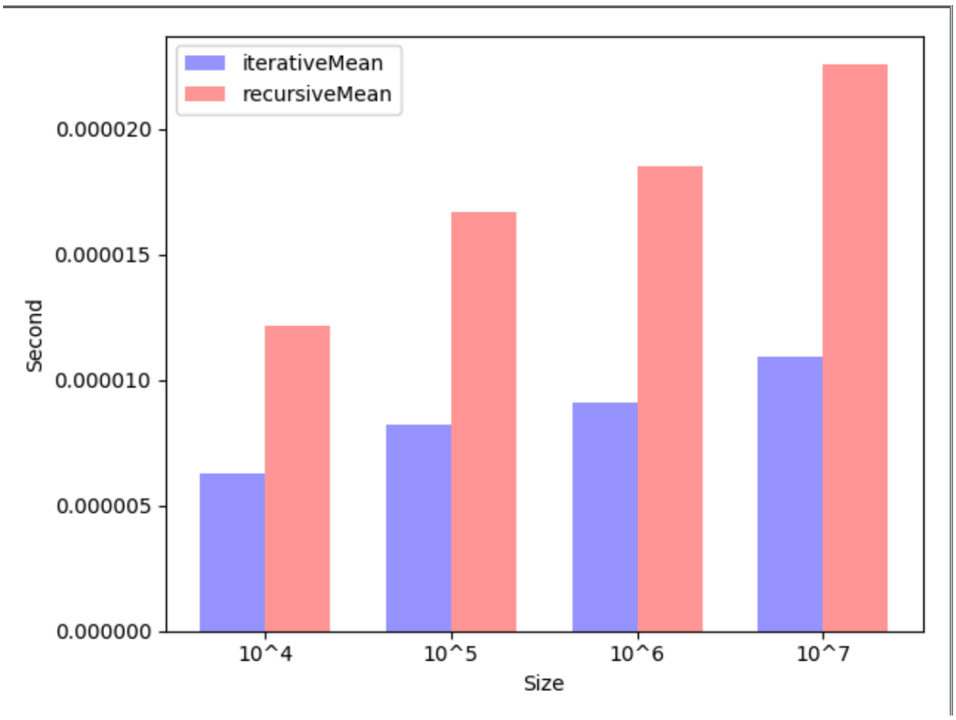
(iv) Experimental results confirm the theoretical results I found in (a) because the graph that i found looks like log base 2 graph.

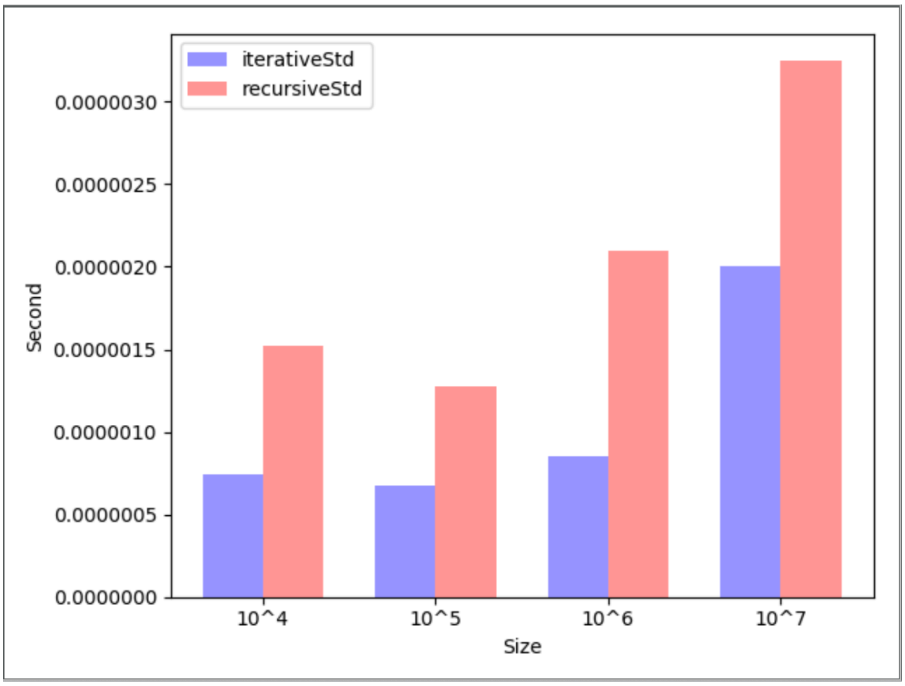


c)(i)

Algorithm	$n = 10^4$		$n = 10^5$		$n = 10^6$		$n = 10^7$	
	μ	σ	μ	σ	μ	σ	μ	σ
Iterative	6.1ms	0.6ms	7.9ms	0.9ms	10.1ms	0,7ms	10.6ms	0.7ms
Recursive	11.8ms	1.9ms	16.2ms	2.1ms	18.8ms	2.1ms	22.8ms	2.2ms

(ii)



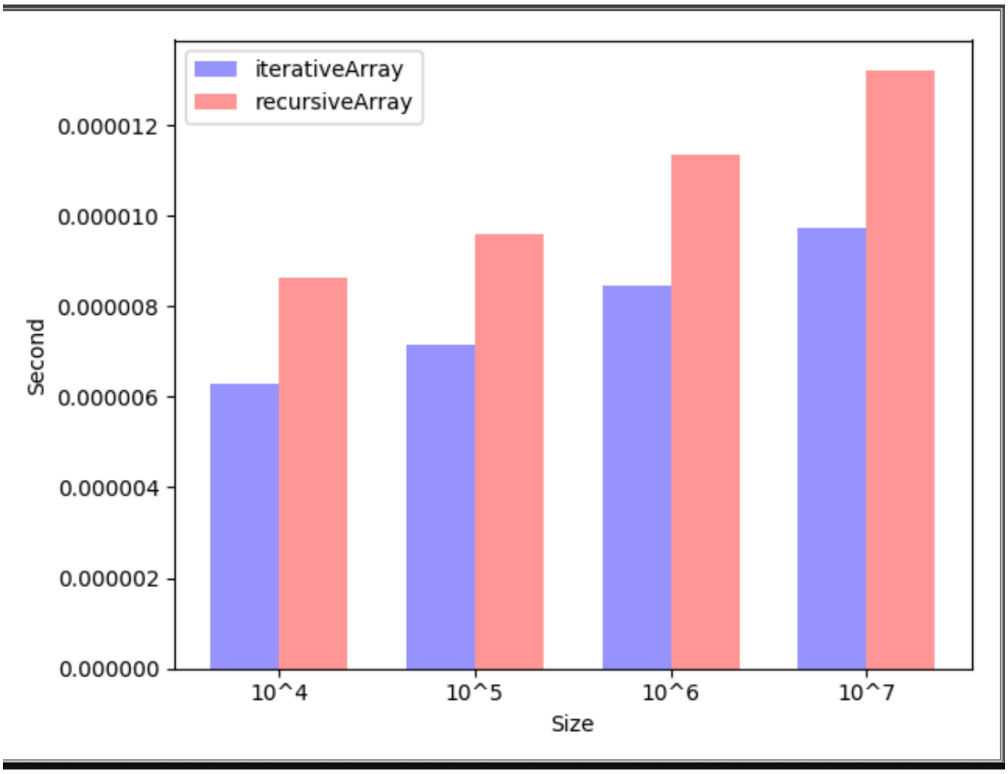


(iii) They are almost same, due to the random key values running times are slightly lower than worst case.

d) I changed the parameters of recursive function call. Now it do not create new arrays, instead it slides over one array.
T(N) = T(N/2) + O(1) is the recurrence relation. T(N/2) comes from the last if else which has recurrence, O(1) comes from len function. If we apply Master's theorem to this recurrence relation, a = 1, b = 2 and n^log (1 base 2) will be n to 0 which is 1. We got the case 2 because f(n) and n^log(a base b) grows at same rates. According to case 2, T(n) = theta(1*logn) which is (logn).
(i)

Algorithm	$n = 10^4$	$n = 10^5$	$n = 10^6$	$n = 10^7$
Iterative	5.5 microsec	6.8 microsec	8.2 microsec	10.3 microsec
Recursive	7.5 microsec	9.0 microsec	11.4 microsec	13.2 microsec

(ii)



(iii) They are almost same after improvement.

(iv) Experimental results confirm the theoretical results i found in part (i) according to graphs.