Efe Şencan 25083

Homework 1

1) Logn = O(n), n = O(log(n!)), (log(n!) = O(nlogn), nlogn = O(n^{100}), n^{100} = O((logn)!), (logn)! = O(2^n), 2^n = $n2^n$, $n2^n$ = O(n!), n! = O($2^{2^{\wedge}n}$)

$\log n = O(n)$ $\log n \leq c.n$ for $c=1$ $n>4$

$n = O(\log n!)$ $n \leq c.\log n!$ $2^n \leq c.n!$ for $c=1$ $n>5$

$\log n! = O(n\log n)$ $\log n! \leq c.n\log n$ $n! \leq n^n.c$ for $c=1$ $n \geq 5$

$n\log n = O(n^{100})$ $n\log n \leq c.n^{100}$ for $c=1$ $n>20$

$n^{100} = O((\log n)!)$ $n^{100} \leq c.(\log n)!$ $\log n = t$, $n = 2^t$ $2^{100t} \leq c.t!$ $c = 2^{100}$ $n>50$

$(\log n)! = O(2^n)$ $(\log n)! = c.2^n$ $\log n = t$, $n = 2^t$ $t! = 2^{c.2^t}$ $c=1$ $n>32$

$2^n = O(n2^n)$ $2^n \leq c.n.2^n$ for $c=1$ $n>1$

$n.2^n = O(n!)$ $n.2^n \leq c.n!$ for $c=1$ $n>10$

$n! = O(2^{2^n})$ $n! = c.2^{2^n}$ for $c=1$ $n>5$

2)

Efe Şencan 25083

2) a) $T(n) = 2T(n/2) + n^3$ (Apply Master Theorem)

$a = 2 \quad b = 2 \quad c = 3 \quad \log_b a = 1 \quad c > \log_b a$

Case 3:

$\Theta(f(n)) = \Theta(n^3)$ //

b) $T(n) = 7T(n/2) + n^2$ (Master Theorem)

$a = 7 \quad b = 2 \quad c = 2 \qquad \log_b a > c$

Case 1:

$\Theta(n^{\log_2 7})$ //

c) $T(n) = 2T(n/4) + \sqrt{n}$ (Master Theorem)

$a = 2 \quad b = 4 \quad c = \frac{1}{2} \quad \log_b a = c$

Case 2:

$f(n) = \Theta(n^{\log_b a} \log^{k+1} n)$ for $k=0$ $T(n) = \Theta(n^c \log n)$

$= \Theta(\sqrt{n} \log n)$ //

d) $T(n) = T(n-1) + n$ (We cannot apply master theorem since $b \neq 1$).

n, $n-1$, $n-2$ $\} O(n^2)$

Apply substitution method.

Prove: $T(n) = O(n^2) \quad T(n) \leq c.n^2 \qquad$ Guess $O(n^2)$

$T(k-1) \leq c(k-1)^2$ for $k < N$

$T(n) \leq \underbrace{c(n-1)^2}_{cn^2 - 2cn + c} + n \qquad 2cn - c - n > 0$

desired $-(2cn - c - n)$ $\qquad$ for $c = 1 \quad n > 1$ //

d) So $O(n^2)$

3)

3) (i) Running time: $(\log n + c) = O(\log n)$

(ii) $T(n) = T(n/2) + \underbrace{O(n)}_{cn}$

Apply Master Theorem:

$a=1 \quad b=2 \quad c=1 \qquad f(n) = \theta(n^c)$ where $c > \log_b a$

Since $c > \log_b a$

Case 3:

$T(n) = \theta(n)$ Upper bound $O(n)$

b)

CPU properties: i7 8750H 2.20GHZ, RAM: 16GB, OS: Windows 10 Pro.

Running time of iterative binary search of size 10^4: 4.638000000056763e-06(s)

Running time of recursive binary search of size 10^4: 3.6638000000976945e-05(s)

Running time of iterative binary search of size 10^5: 6.493000000773463e-06(s)

Running time of recursive binary search of size 10^5: 0.00036915900000016731(s)

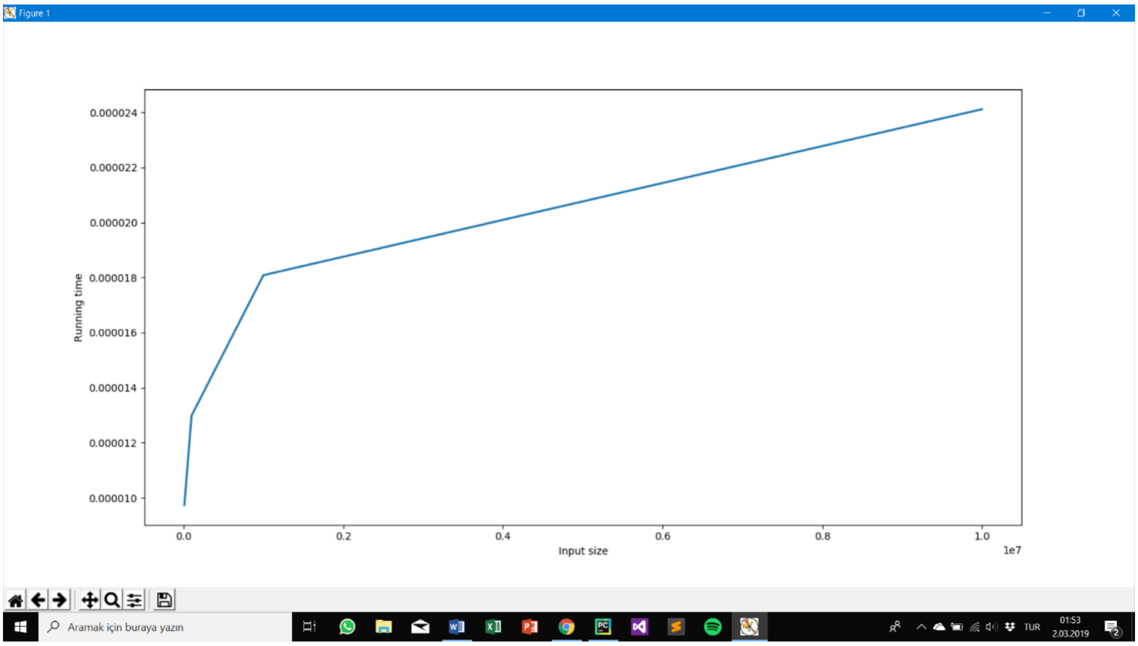
Running time of iterative binary search of size 10^6: 8.812000000801845e-06(s)

Running time of recursive binary search of size 10^6: 0.018551196999997188(s)

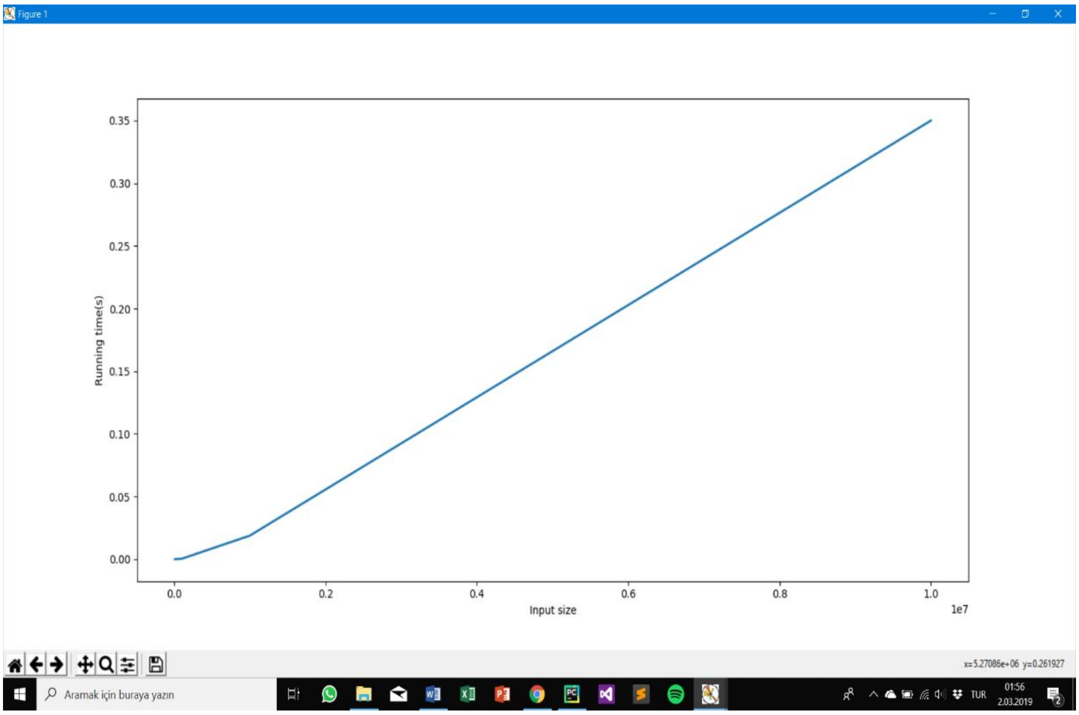
Running time of iterative binary search of size 10^7: 1.0666999997965831e-05(s)

Running time of recursive binary search of size 10^7: 0.3560637009999965(s)

Efe Şencan 25083



(Iterative binary search graph with respect of input size and running time(s))



(Recursive binary search graph with respect of input size and running time(s))

iii) When we look at the results of the running times and the graphs, we can see that, the running time of the iterative binary search algorithm increases logarithmically (logn) as the input size increases which we also showed theoretically. While the running time of the recursive binary search algorithm increases more rapidly(O(n) linearly) as the input size increases. As a result, iterative algorithm works faster than the recursive one for that case and it is more scalable because after some point recursive one will take much longer time to search for the key value.

iv) Our experimental results confirm the theoretical results we found in part a. Because we thought that, the time complexity of the iterative binary search would be logarithmic, and when we plot its graph, we obtain a logarithmic graph. Also, we found that, the time complexity of the recursive binary search algorithm would be O(n), and our graph also confirms this complexity with respect to running time and input size. The running times (seconds) also supports the result we found theoretically.

c)

iii) The average running time of iterative binary search of size 10^ 4 : 8.996600000088506e-07(s)

The average running time of recursive binary search of size 10^ 4 : 3.0219219999998436e-05(s)

The standard deviation of the running time of iterative binary search of size 10^ 4 3.430741343149457e-07(s)

The standart deviation of the running time of recursive binary search of size 10^ 4 1.7835091217460157e-06(s)

The average running time of iterative binary search of size 10^ 5 : 1.094400000005713e-06(s)

The average running time of recursive binary search of size 10^ 5 : 0.0003409164999999925(s)

The standart deviation of the running time of iterative binary search of size 10^ 5 5.68151208678278e-07(s)

The standart deviation of the running time of recursive binary search of size 10^ 5 5.504632817881058e-05(s)

The average running time of iterative binary search of size 10^ 6 : 4.044080000022987e-06(s)

The average running time of recursive binary search of size 10^ 6 : 0.02007461112000005(s)

The standart deviation of the running time of iterative binary search of size 10^ 6 5.546470412923955e-07(s)

The standart deviation of the running time of recursive binary search of size 10^ 6 0.0005527108627479612(s)

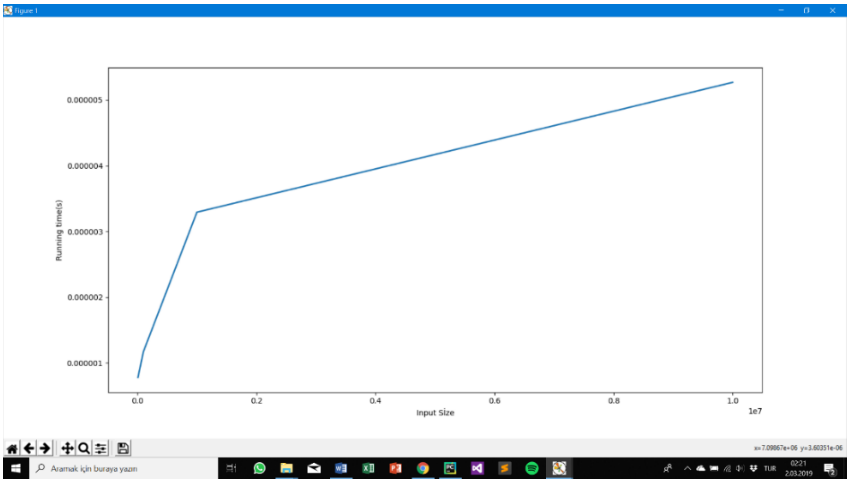
The average running time of iterative binary search of size 10^ 7 : 6.9658199994648835e-06(s)

The average running time of recursive binary search of size 10^ 7 : 0.5387107696599989(s)

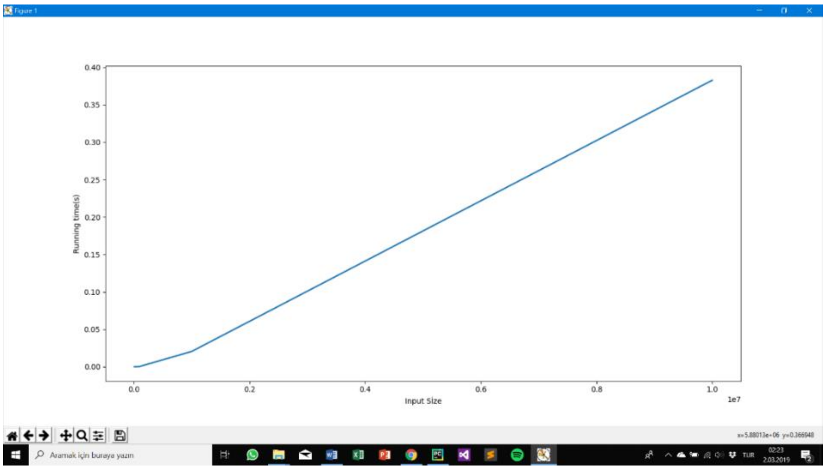
The standart deviation of the running time of iterative binary search of size 10^ 7 1.3593716833572996e-06(s)

The standart deviation of the running time of recursive binary search of size 10^ 7 0.05675364567035283(s)

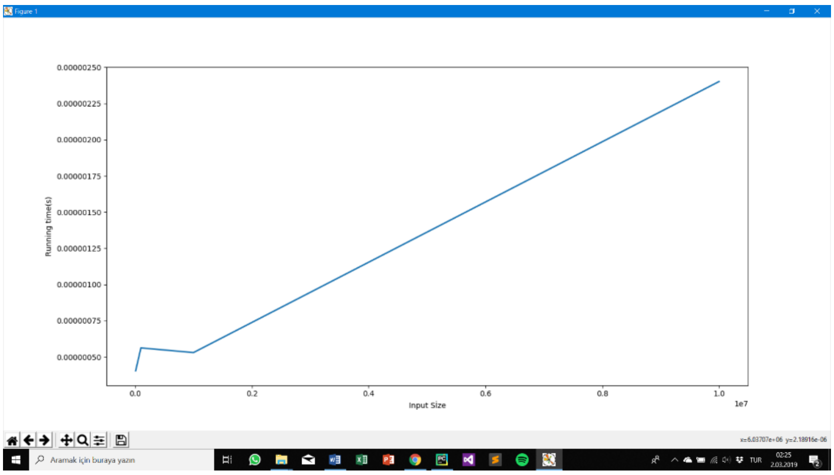
Efe Şencan 25083



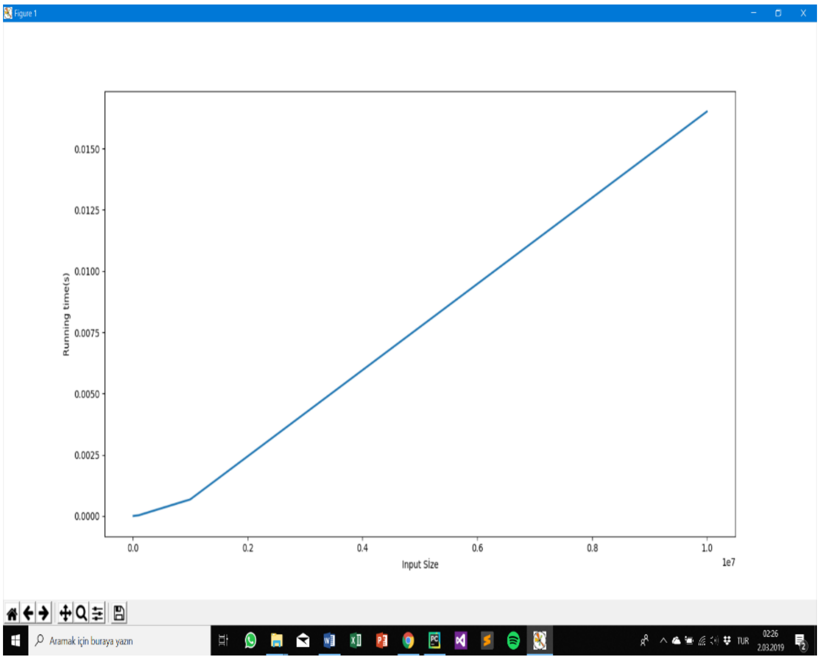
(average running time of iterative binary search versus input size)



(average running time of recursive binary search versus input size)



(standart deviation of iterative binary search)



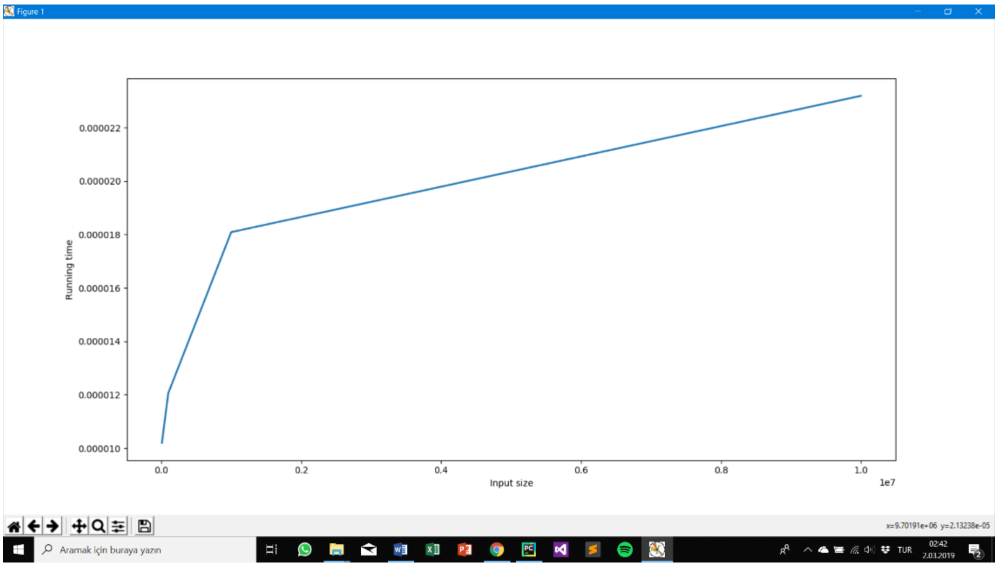
(standart deviation of the recursive binary search)

iii) In part b, we were trying to find the number “1” in our list, which was the worst case. On the other hand, this time since we are trying to search for a number that is randomly generated, our average results with respect to the running time, is less than we observed in part b most of the time.

d) We can improve the time complexity of the recursive binary search algorithm from O(n) to O(logn) by forming variables called start (which corresponds to the starting index of the array) and end (which corresponds to the last index of the array). At each iteration we can check the value of the key value (the one we search) whether it is higher or lower than the middle element of the current array, if it is lower, than we update or last index to the value of the middle element's index, if not, we assign the start value to the index of the middle element +1. Thus, we do not need to use list slicing method which costs O(n). Instead we get constant time complexity from there which results O(logn) overall.

The code of improved recursive binary search:

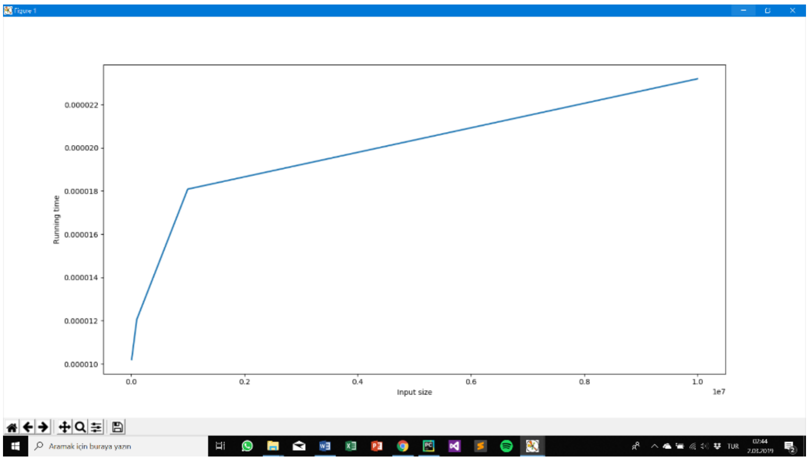
```
def improvedRecursive(item,arr,start,end):
    if(last == first):
        return arr[first] == item
    if(start > end):
        return False
    mid = (start + end) // 2
    if(arr[mid] > item):
        end  = mid
        return improvedRecursive(item,arr,start,end)
    elif(arr[mid] < item):
        start = mid + 1
        return improvedRecursive(item, arr, start, end)
    else:
        return arr[mid] == item
```



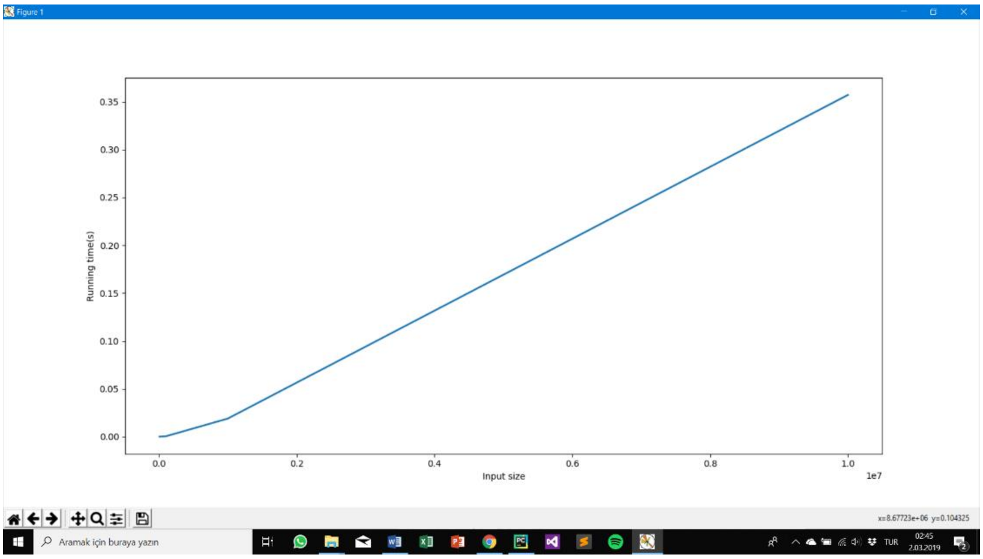
(improved recursive binary search)

As we can observe from the graph, the function becames logarithmic(logn) instead of linear after the improvements we made.

Efe Şencan 25083



(iterative binary search)



(recursive binary search)

The results of the improved recursive binary search by the seconds:

Improved recursive binary search for size 10^ 4 : 8.812000000024689e-06(s)

Improved recursive binary search for size 10^ 5 : 1.1595000000030886e-05(s)

Improved recursive binary search for size 10^ 6 : 1.437700000028741e-05(s)

Improved recursive binary search for size 10^ 7 : 2.365200000298273e-05(s)

Source Code:

```
from timeit import default_timer as timer
import random
import copy
import matplotlib.pyplot as plt

import scipy as sp
from scipy.stats import t
from numpy.polynomial.polynomial import polyfit


def iterBinarySearch(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


def recursiveBinarySearch(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 recursiveBinarySearch(alist[:midpoint],item)
            else:
                return recursiveBinarySearch(alist[midpoint+1:],item)



def improvedRecursive(item,arr,start,end):
    if(end == start):
        return arr[start] == item
    if(start > end):
        return False
    mid = (start + end) // 2
    if(arr[mid] > item):
        end  = mid
        return improvedRecursive(item,arr,start,end)
    elif(arr[mid] < item):
        start = mid + 1
        return improvedRecursive(item, arr, start, end)
    else:
        return arr[mid] == item


power = 4
improved_timelist = []

print("Improved recursive started to work")
for i in range(0,4):
    liste = []
    result_improve = 0
    for i in range(0,pow(10,power)):
        liste.append(random.randint(1,pow(10,7)))
    liste.sort()
    start_improve = timer()
    improvedRecursive(1,liste,0,len(liste) - 1)
    end_improve = timer()
    result_improve = end_improve - start_improve
    print("Improved recursive binary search for size 10^",power,":
",result_improve)
    improved_timelist.append(result_improve)
    power += 1
```

```
iter_time_list = []
recursive_time_list = []

first_list = []
for i in range(0,pow(10,4)):
    first_list.append(random.randint(1,pow(10,7)))

print("**********************")
first_result_iter = 0.0
first_result_recursive = 0.0
first_list.sort()
first_start_iter = timer()
iterBinarySearch(first_list,1)
first_end_iter = timer()
first_result_iter += first_end_iter - first_start_iter
print("Iterative binary search time for size 10^4: ", first_result_iter)
iter_time_list.append(first_result_iter)
first_start_recursive = timer()
recursiveBinarySearch(first_list,1)
first_end_recursive = timer()
first_result_recursive += first_end_recursive - first_start_recursive
print("Recursive binary search time for size 10^4: ",
first_result_recursive)
recursive_time_list.append(first_result_recursive)
print("**********************")


second_list = []
for i in range(0,pow(10,5)):
    second_list.append(random.randint(1,pow(10,7)))

second_result_iter = 0.0
second_result_recursive = 0.0
second_list.sort()
second_start_iter = timer()
iterBinarySearch(second_list,1)
second_end_iter = timer()
second_result_iter += second_end_iter - second_start_iter
print("Iterative binary search time for size 10^5: ",second_result_iter)
iter_time_list.append(second_result_iter)
second_start_recursive = timer()
recursiveBinarySearch(second_list,1)
second_end_recursive = timer()
second_result_recursive += second_end_recursive - second_start_recursive
print("Recursive binary search time for size 10^5: ",
second_result_recursive)
recursive_time_list.append(second_result_recursive)

print("***************************")

third_list = []
for i in range(0,pow(10,6)):
    third_list.append(random.randint(1,pow(10,7)))

third_result_iter = 0.0
third_result_recursive = 0.0
third_list.sort()
third_start_iter = timer()
iterBinarySearch(third_list,1)
third_end_iter = timer()
third_result_iter += third_end_iter - third_start_iter
```

```
print("Iterative binary search time for size 10^6: ",third_result_iter)
iter_time_list.append(third_result_iter)
third_start_recursive = timer()
recursiveBinarySearch(third_list,1)
third_end_recursive = timer()
third_result_recursive += third_end_recursive - third_start_recursive
print("Recursive binary search time for size 10^6: ",
third_result_recursive)
recursive_time_list.append(third_result_recursive)
print("*********************")

fourth_list = []
for i in range(0,pow(10,7)):
    fourth_list.append(random.randint(1,pow(10,7)))

fourth_result_iter = 0.0
fourth_result_recursive = 0.0
fourth_list.sort()
fourth_start_iter = timer()
iterBinarySearch(fourth_list,1)
fourth_end_iter = timer()
fourth_result_iter += fourth_end_iter - fourth_start_iter
print("Iterative binary search time for size 10^7: ",fourth_result_iter)
iter_time_list.append(fourth_result_iter)
fourth_start_recursive = timer()
recursiveBinarySearch(fourth_list,1)
fourth_end_recursive = timer()
fourth_result_recursive += fourth_end_recursive - fourth_start_recursive
print("Recursive binary search time for size 10^7: ",
fourth_result_recursive)
recursive_time_list.append(fourth_result_recursive)

print("***********************")


input_size = [pow(10,4),pow(10,5),pow(10,6),pow(10,7)]
plt.plot(input_size,iter_time_list,linewidth = 2.0)
plt.xlabel("Input SİZe")
plt.ylabel("Running time(s)")
plt.show()

plt.plot(input_size,recursive_time_list,linewidth = 2.0)
plt.xlabel("Input size")
plt.ylabel("Running time(s)")
plt.show()


keys_iter_timelist = []
keys_recursive_timelist = []
power = 4
result = 0
for i in range(0,4):
    liste = []
    for i in range(0,pow(10,power)):
        liste.append(random.randint(1,pow(10,7)))
    liste.sort()
    for x in range(0,50):
        number = random.randint(1,pow(10,7))
        start_time = timer()
        iterBinarySearch(liste,number)
        end_time = timer()
```

```
    result = end_time - start_time
    power += 1
    keys_iter_timelist.append(result/50)

print(improved_timelist)
plt.plot(input_size,improved_timelist,linewidth = 2.0)
plt.xlabel("Input size")
plt.ylabel("Running time")
plt.show()
```

Part b:

```
power = 4
for i in range(0,4):
    add = []
    second = []
    result = 0
    second_result = 0
    liste = []
    for i in range(0,pow(10,power)):
        liste.append(random.randint(1,pow(10,7)))
    liste.sort()
    for x in range(0,50):
        number = random.randint(1,pow(10,7))
        start_time = timer()
        iterBinarySearch(liste,number)
        end_time = timer()
        add.append(end_time - start_time)
        second_start_time = timer()
        recursiveBinarySearch(liste,number)
        second_end_time = timer()
        second.append(second_end_time - second_start_time)
        second_result += second_end_time - second_start_time
        result += end_time - start_time

    keys_recursive_timelist.append(second_result/50)
    keys_iter_std.append(stdev(add))
    keys_recursive_std.append(stdev(second))
    keys_iter_timelist.append(result/50)
    print("The average running time of iterative binary search of size 
10^",power,": ",result/50)
    print("The average running time of iterative binary search of size 
10^",power,": ",second_result/50)
    print("The standart deviation of the running time of iterative binary 
search of size 10^",power,stdev(add))
    print("The standart deviation of the running time of recusive binary 
search of size 10^", power, stdev(second))
    power+=1



input_size = [pow(10,4),pow(10,5),pow(10,6),pow(10,7)]
plt.plot(input_size,keys_iter_timelist,linewidth = 2.0)
plt.xlabel("Input Sİze")
plt.ylabel("Running time(s)")
```

```
plt.show()



plt.plot(input_size,keys_recursive_timelist,linewidth = 2.0)
plt.xlabel("Input Size")
plt.ylabel("Running time(s)")
plt.show()


plt.plot(input_size,keys_iter_std,linewidth = 2.0)
plt.xlabel("Input Size")
plt.ylabel("Running time(s)")
plt.show()


plt.plot(input_size,keys_recursive_std,linewidth = 2.0)
plt.xlabel("Input Size")
plt.ylabel("Running time(s)")
plt.show()
```