Homework #2

Sorting data and functions

Anastasiia Konoplina

---

1. Measure the actual speed of binary search. How many searches can you perform in one minute for various sizes of the arrays?

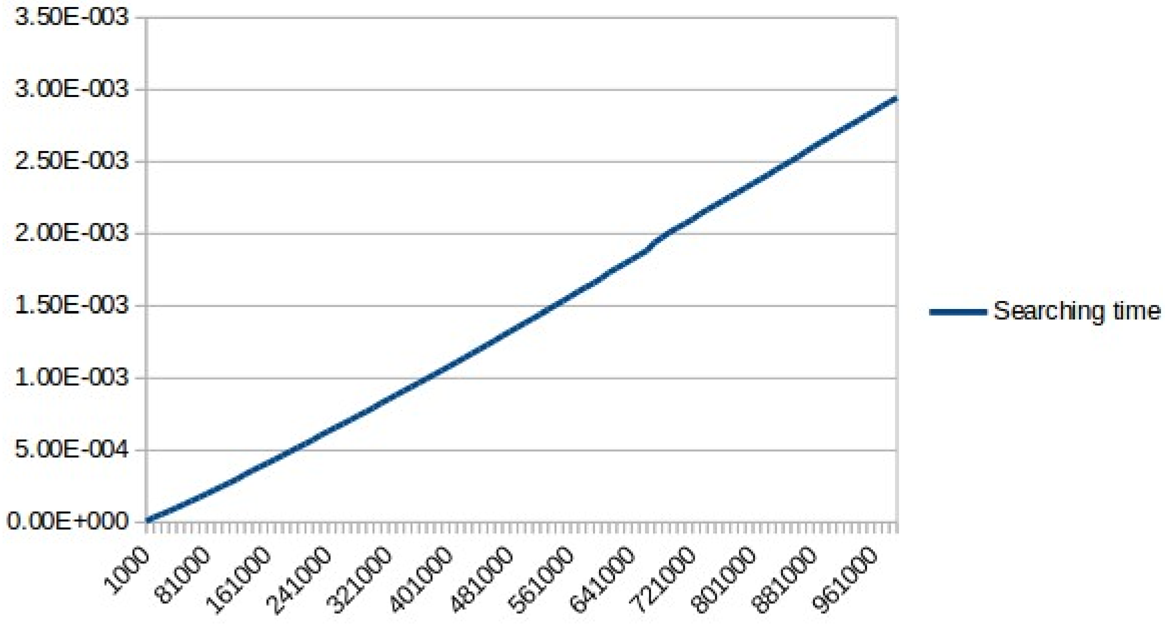
I have not found built-in binary search in python, so I created it manually. The code is below:

```
def binarySearch(array, value):

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

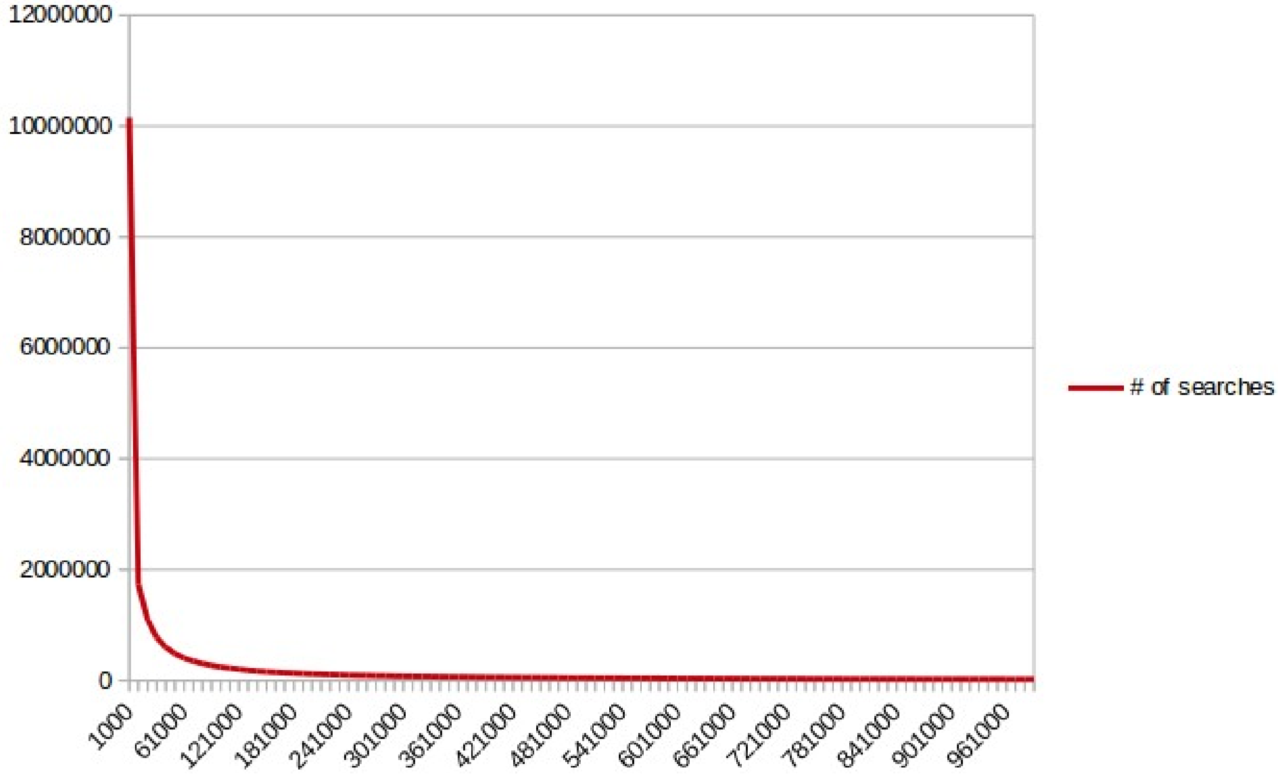
I have created arrays starting from array of 1000 elements and iteratively increased size of array by 10 000 up to 1000000. For each size of array I made 10 measures and calculated mean. The code is below:

```
for i in range(1000, 1000000, 10000):

    test_array = [random.randint(1,500) for _ in range(i)]

    test_array = sorted(test_array)

    for k in range(1,10):

        start_time = time.time()

        res = binarySearch(test_array, 20)

        res_time = time.time() - start_time

        time_array.append(res_time)

    res_time = sum(time_array) / float(len(time_array))

    search_time.append(res_time)
```

The chart of binary search time based on this information is below:



The chart of array size – number of searches in 1 minute dependency is below:



---

2. Implement the Quicksort with two pivots. Describe how you might try to avoid bad splits to make your code "unbreakable".

My implementation of dual Quicksort is below:

```python
def dualQuickSort(array):

    if len(array) > 2:
        a = random.sample(array,2)
        p1 = min(a)
        p2 = max(a)
        part1 = [i for i in array if i <= p1]
        part2 = [i for i in array if ((i > p1) & (i < p2))]
        part3 = [i for i in array if i >= p2]
        part1 = dualQuickSort(part1)
        part2 = dualQuickSort(part2)
        part3 = dualQuickSort(part3)
    else:
        return array
    res = part1 + part2 + part3
    return res
```

For test I used array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1], where most of elements are not unique. So that when I randomly choose 2 pivots – they might be the same. To fix this problem I used random.sample() function, which takes only unique elements from population.

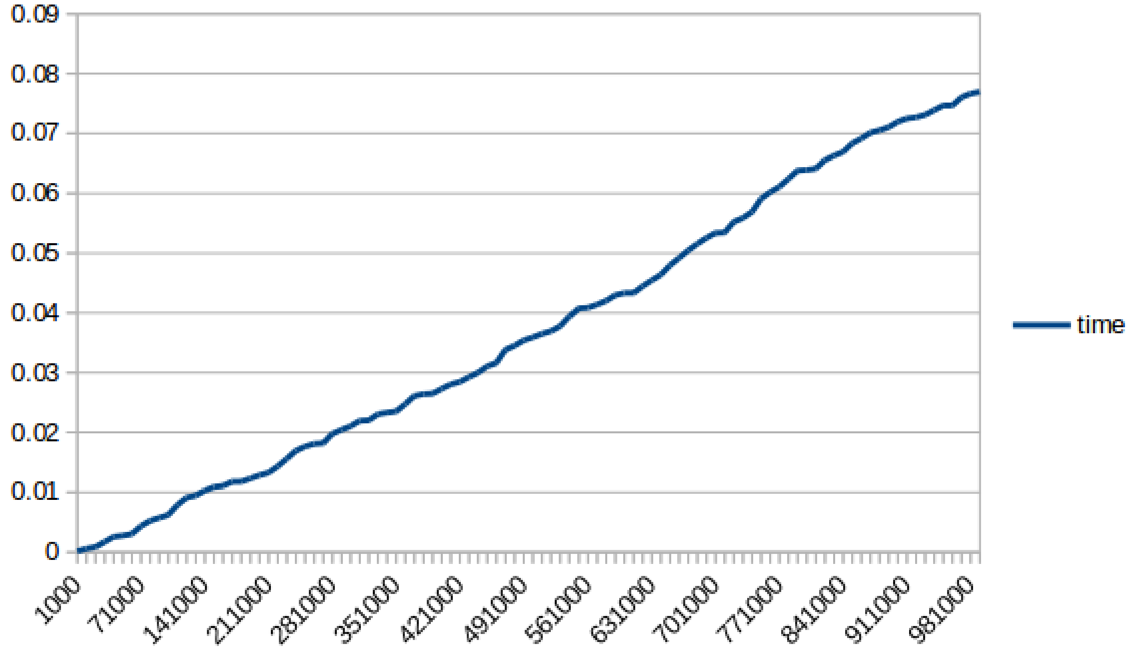
---

3. Implement the linear-time order statistics calculations on the unsorted data, measure the speed. Run many searches - does the speed improve over time? (does the data become sorted slowly?)

I have chosen Quickselect algorithm, its implementation is below:

```python
def quickSelect(array, k):

    p = array[(len(array) // 2)]
    part1 = [i for i in array if i < p]
    part2 = [i for i in array if i > p]
    p1 = len(part1)
    r = len(array) - len(part1) - len(part2)
    if k >= p1 and k < p1 + r:
        return ppart1 = [i for i in array if i < p]
    elif m > k:
        return quickSelect(part1, k)
    else:
        return quickSelect(part2, k - p1 - r)
```

To prove that in average this is linear function, I ran it for arrays [1000; 1000000] elements with 10 000 step, and took mean of 10 runs of every array. The chart of array size – time dependency is below:



4-5. Solve task 3-3 from CLRS - order functions by their growth rate. Group together those that belong to the same Theta() group.

Honestly, I have no idea how to order them, I mean I could not do it by myself, that is why I tried to understand solutions I could find on stackoverflow and so on:

To order these functions 2 statements (do not remember where I took it from) should be used:

- Exponential functions grow faster than polynomial functions, which grow faster than polylogarithmic functions.
- The base of a logarithm doesn't matter asymptotically, but the base of an exponential and the degree of a polynomial do matter.

They used several math formulas (identities):

- (lgn)^lgn = n^lglgn
- 4^lgn = n^2
- 2^lgn = n
- 2 = n^(1/lgn)
- 2^sqr(2lgn) = n^sqr(2/lgn)
- sqr(2)^2lgn = sqr(n)

- lg*(lgn) = (lg*n) – 1

The ordered list of functions is below:

1. 1
2. lg(lg* n)
3. lg* n and lg*(lg n)
4. 2^lg* n
5. ln ln l
6. sqr(lg n)
7. ln n
8. lg^2 n
9. 2^sqr(2lg n)
10. sqr(2)^lg n
11. n
12. n lgn
13. n^2
14. n^3
15. n^lglgn
16. (3/2)^n
17. 2^n
18. n2^n
19. e^n
20. n!
21. (n+1)!
22. 2^2^n
23. 2^2^(n+1)

1) $e^n = 2^n \left(\frac{e}{2}\right)^n = \omega(n 2^n)$

because $\left(\frac{e}{2}\right)^n = \omega(n)$

2) $(\lg n)! = \omega(n^3)$

$\lg (\lg n)! = \Theta(\lg n \lg \lg n)$

$\lg(n^3) = 3 \lg n$

$\lg \lg n = \omega(3)$

3) $(\sqrt{2})^{\lg n} = \omega(2^{\sqrt{2 \lg n}})$

$\lg (\sqrt{2})^{\lg n} = \frac{1}{2} \lg n$

$\lg 2^{\sqrt{2 \lg n}} = \sqrt{2 \lg n}$

$\frac{1}{2} \lg n = \omega(\sqrt{2 \lg n})$

4) $2^{\sqrt{\lg n}} = \omega(\lg^2 n)$

$\lg 2^{\sqrt{2 \lg n}} = \sqrt{2 \lg n}$

$\lg \lg^2 n = 2 \lg \lg n$

$\sqrt{2 \lg n} = \omega(2 \lg \lg n)$

5) $\ln \ln n = \omega(2^{\lg^* n})$

$\lg 2^{\lg^* n} = \lg^* n$

$\lg \ln \ln n = \omega(\lg^* n)$

6) $\lg(n!) = \Theta(n \lg n)$

7) $n! = \Theta(n^{n+1/2} e^{-n})$

8) $(\lg n)! = \Theta((\lg n)^{\lg n + 1/2} e^{-\lg n})$