MATH 182: HOMEWORK 2

Due Friday Oct 14 at 12pm.

Written answer

Submit your solutions to "Homework 2 written" on Gradescope. Make sure to provide a full justification (err on the side of "too detailed").

Problem 1. Find the time complexity of $T(n)$ satisfying $T(n) = 9T(n/3) + O(n^2)$.

Problem 2. Find the time complexity of $T(n)$ satisfying $T(n) = 50T(n/3) + O(n^3)$.

Problem 3. Find the time complexity of $T(n)$ satisfying $T(n) = 10T(n/4) + O(n^2)$.

Problem 4. Find the time complexity of the following function.

```
# A is an integer array
def foo(A):
    n = len(A)
    if n <= 1:
        print("hello")
    else:
        for i in range(n):
            for j in range(i):
                print(j)
                print(i)
        foo(A[: n//2])
        foo(A[n//4 : 3*n//4])
        foo(A[n//2:])
```

Programming problem

Submit to "Homework 2 Programming" the following:

- The file(s) containing your code.
- The file `submission_a.txt`
- The file `submission_b.txt`
- The file `submission_c.txt`
- The file `submission_d.txt`

We will implement an $O(n)$ algorithm to compute the median of an array. By definition, the **median** of an array A is the `len(A)//2`-th smallest element of A (so if A has even length, you take the larger of the middle two).

Actually, we will have to do something more general. We will implement an $O(n)$ algorithm to find the i-th smallest element of an array.

Do NOT use a built-in median algorithm or sorting algorithm for this problem.

Problem A. Write an $O(1)$ implementation of the function `median_of_few`, which takes in a nonempty integer array A of length at most 5, and returns its median.

Apply `median_of_few` to each row of `data_a.txt`, and output each result as a row of `submission_a.txt`, as in the sample below.

For instance, if `data_a.txt` was the following:

```
[20, 20, 62, 67]
[72, 20, 54, 3]
[72, 52, 15]
[56]
[37, 20, 20, 84, 12]
```

Then `submission_a.txt` would be the following:

```
62
54
52
56
20
```

Problem B. Write an $O(n)$ implementation of the function `medians_of_quintets`, which takes in an integer array A, and returns an array consisting of the median of the first five entries, followed by the median of the next five entries, followed by the median of the next five entries, and so on (the last median might be the median of less than five entries). Calculate all medians with `median_of_few`.

Apply `medians_of_quintets` to each row of `data_b.txt`, and output each result as a row in `submission_b.txt`, as in the sample below.

For instance, if `data_b.txt` was the following:

```
[48, 63, 62, 8, 71, 86, 58, 98, 86]
[18, 34, 91, 68, 3, 23, 26, 79, 60, 83, 3, 84, 90, 40, 40, 30, 44, 86]
[28, 4, 26, 6, 23, 71, 18, 45]
[62, 46, 83, 24, 66]
[28, 29]
[45]
```

Then `submission_b.txt` would be the following:

```
[62, 86]
[34, 60, 40, 44]
[23, 45]
[62]
[29]
[45]
```

Problem C. Write an $O(n)$ implementation of the function `partition_at`, which takes in an integer array A and an integer k, and returns a tuple $(A_<, A_>)$ of two arrays, where $A_<$ is the subsequence of entries of A which are less than k, and $A_>$ is the subsequence of entries of A which are greater than k.

Apply `partition_at` to each row of `data_c.txt`, and output each result as a row in `submission_c.txt`, as in the sample below.

For instance, if `data_c.txt` was the following:

```
([72, 4, 88, 85, 1, 32, 52, 88, 19, 21, 83], 3)
([70, 11, 88, 77, 6, 77, 34, 14, 31, 62, 38, 10, 95], 63)
([80, 62, 51, 96, 57, 40, 65, 63, 83, 70, 95, 78, 91, 85, 26, 34], 74)
([17, 80, 56, 8, 31, 36], 15)
([6, 5, 4, 3], 5)
([4, 5, 4, 5], 4)
([0, 1, 2, 1, 0], 2)
([4, 4, 4], 4)
```

Then `submission_c.txt` would be the following:

```
([1], [72, 4, 88, 85, 32, 52, 88, 19, 21, 83])
([11, 6, 34, 14, 31, 62, 38, 10], [70, 88, 77, 77, 95])
([62, 51, 57, 40, 65, 63, 70, 26, 34], [80, 96, 83, 95, 78, 91, 85])
([8], [17, 80, 56, 31, 36])
([4, 3], [6])
([], [5, 5])
([0, 1, 1, 0], [])
([], [])
```

Problem D. Let `quickselect` be the function which takes in a nonempty integer array A and a natural number i less than $\text{len}(A)$, and returns the i-th smallest element of A.

Write the following implementation of `quickselect`. Let $n = \text{len}(A)$.

- If A has length 1, then return $A[0]$.
- Otherwise, find the median m of `medians_of_quintets(A)` by recursively calling `quickselect`, and run `partition_at` on A and m to get $(A_<, A_>)$. If $A_<$ is long enough so that the desired element is in $A_<$, recursively call `quickselect` on $A_<$ with the appropriate index. If $A_>$ is long enough so that the desired element is in $A_>$, recursively call `quickselect` on $A_>$ with the appropriate index. Otherwise, m is the desired element, so return m.

Apply `quickselect` to each row of `data_d.txt`, and output each result as a row in `submission_d.txt`, as in the sample below.

For instance, if `data_d.txt` was the following:

```
([78, 3, 96, 25], 2)
([21, 59, 85, 53, 6, 10, 69, 84], 0)
([91, 79, 41, 16, 41, 41, 56, 27, 36, 63, 97, 70, 98, 78, 52, 60, 20, 69], 4)
([64, 20, 45, 57, 84, 76, 97, 31, 70, 5, 64, 75, 94, 93, 2], 3)
([29], 0)
([28, 28, 1, 30], 1)
```

Then `submission_d.txt` would be the following:

```
78
6
41
```

```
31
29
28
```