

Problem Set 1

This problem set is divided into two parts: Part A problems are theory questions, and Part B problems are programming tasks.

Part A questions are due **Tuesday, September 22nd at 11:59PM**.

Part B questions are due **Thursday, September 24th at 11:59PM**.

Solutions should be turned in through the course website in PDF form using \LaTeX or scanned handwritten solutions.

A template for writing up solutions in \LaTeX is available on the course website.

Remember, your goal is to communicate. Full credit will be given only to the correct solution which is described clearly. Convolved and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A: Due Tuesday, September 22nd

1. (18 points) Asymptotic Growth

For each group of six functions below, rank the functions by increasing order of growth; that is, find an arrangement g_1, g_2, \dots, g_6 of the functions satisfying $g_1 = O(g_2)$, $g_2 = O(g_3)$, \dots , $g_5 = O(g_6)$. Partition each list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$.

(a) (6 points) Group 1:

$$\begin{aligned} f_1(n) &= 10^{6006}n^2, & f_2(n) &= 1/n, & f_3(n) &= n^{6.006} \\ f_4(n) &= n, & f_5(n) &= 1/6.006, & f_6(n) &= 6.006n \end{aligned}$$

(b) (6 points) Group 2:

$$\begin{aligned} f_1(n) &= n \log n, & f_2(n) &= \log(\log(n)), & f_3(n) &= \binom{n}{50} \\ f_4(n) &= n^{1/50}, & f_5(n) &= \log(n), & f_6(n) &= \log(\sqrt{10n}) \end{aligned}$$

(c) (6 points) Group 3:

$$\begin{aligned} f_1(n) &= 2^n, & f_2(n) &= n2^n, & f_3(n) &= 2^{n+1} \\ f_4(n) &= 4^n, & f_5(n) &= n!, & f_6(n) &= 3^{\sqrt{n}} \end{aligned}$$

2. (12 points) Binary Search

Binary search is a fast algorithm used for finding membership of an element in a sorted list. The iterative version of the algorithm is given below. The function takes a sorted list of numbers, `alist`, and a query, `item`, and returns `true` if and only if `item ∈ alist`. Let n denote the length of the list `alist`.

```

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

```

- (a) (5 points) What is the runtime of the iterative version in terms of n , and why?
 (b) (7 points) Write a concise proof of correctness for the algorithm.

3. (20 points) Uncoordinated Peak Finding

Consider the Peak finding problem discussed in lecture (also refer to Problem 1 in Part B). An excited 6.006 student comes up with the following algorithm to solve the problem:

- 1 Let $n = \text{len}(\text{row}(B))$. Find the maximum element of the $(n/2)^{\text{th}}$ column and call it $c_{max} = B[i][n/2]$
- 2 If $c_{max} \geq B[i][n/2 - 1]$ and $c_{max} \geq B[i][n/2 + 1]$ **return** c_{max}
- 3 If $c_{max} < B[i][n/2 - 1]$ **then** $B = B[1..n][1..n/2-1]$ **else** $B = B[1..n][n/2+1..n]$
- 4 Find the maximum element of the $(n/2)^{\text{th}}$ row of B and call it $r_{max} = B[n/2][j]$
- 5 If $r_{max} \geq B[n/2 - 1][j]$ and $r_{max} \geq B[n/2 + 1][j]$ **return** r_{max}
- 6 If $r_{max} < B[n/2 - 1][j]$ **then** $B = B[1..n/2-1][1..n/2-1]$ **else** $B = B[n/2+1..n][1..n/2-1]$
- 7 **goto** Step 1.

- (a) (10 points) Give a counterexample (an instance of B of size less than 7×7) where the above algorithm fails to find a peak in B even though it exists.
 (b) (10 points) We can fix the above algorithm by keeping track of the running maximum run_{max} as below. Explain how it solves the problem in the previous counterexample.

```

1   $run_{max} = -\infty$ 
2  Let  $n = \text{len}(\text{row}(B))$ . Find the maximum element of the  $(n/2)^{th}$  column and
   call it  $c_{max} = B[i][n/2]$ 
3  If  $c_{max} \geq run_{max}$  then
4     If  $c_{max} \geq B[i][n/2 - 1]$  and  $c_{max} \geq B[i][n/2 + 1]$  then return  $c_{max}$ 
5     If  $c_{max} < B[i][n/2 - 1]$  then  $run_{max} = B[i][n/2 - 1]$  else  $run_{max} =$ 
       $B[i][n/2 + 1]$ 
6     If  $c_{max} < B[i][n/2 - 1]$  then  $B = B[1..n][1..n/2-1]$  else  $B = B[1..n][n/2+1..n]$ 
7 Else /* Update B to be the partition of B containing  $run_{max}$  */
8     If  $run_{max} \in B[1..n][1..n/2-1]$  then  $B = B[1..n][1..n/2-1]$  else  $B = B[1..n][n/2+1..n]$ 
9 Find the maximum element of the  $(n/2)^{th}$  row of B and call it  $r_{max} =$ 
       $B[n/2][j]$ 
10 If  $r_{max} \geq run_{max}$  then
11     If  $r_{max} \geq B[n/2 - 1][j]$  and  $r_{max} \geq B[n/2 + 1][j]$  return  $r_{max}$ 
12     If  $r_{max} < B[n/2 - 1][j]$  then  $run_{max} = B[n/2 - 1][j]$  else  $run_{max} =$ 
       $B[n/2 + 1][j]$ 
13     If  $r_{max} < B[n/2 - 1][j]$  then  $B = B[1..n/2-1][1..n/2]$  else  $B = B[n/2+1..n][1..n/2]$ 
14 Else /* Update B to be the partition of B containing  $run_{max}$  */
15     If  $run_{max} \in B[1..n/2][1..n/2 - 1]$  then  $B = B[1..n/2][1..n/2-1]$  else  $B =$ 
       $B[n/2+1..n][1..n/2-1]$ 
16 goto Step 2.

```

Part B: Due Thursday, September 24th

1. (50 points) Peak Finding

Consider an array A containing n integers. We define a *peak* of A to be an x such that $x = A[i]$, for some $0 \leq i < n$, with $A[i - 1] \leq A[i]$ and $A[i] \geq A[i + 1]$. In other words, a peak x is greater than or equal to its neighbors in A (for boundary elements, there is only one neighbor). Note that A might have multiple peaks.

As an example, suppose $A = [10, 6, 4, 3, 12, 19, 18]$. Then A has two peaks: 10 and 19.

Note that the absolute maximum of A is always a peak, but it requires $\Omega(n)$ time to compute.

- (20 points) Write `quick_find_1d_peak` to compute any peak of array A in $O(\log(n))$ time using the algorithm described in the lecture.

Now consider a two dimensional matrix B of integers of size $n \times n$. We define neighborhood of an element $x = B[i][j]$ as $B[i + 1][j]$, $B[i - 1][j]$, $B[i][j + 1]$ and $B[i][j - 1]$. For elements at the boundary, we consider only three neighbors and for

elements on the 4 corners, only two neighbors are considered. x is defined to be a peak of B if and only if it is greater than or equal to all of its neighbours. Note that the maximum element of B is a possible solution for x but that requires $\Omega(n^2)$ time. For python coding help, the $O(n \log(n))$ algorithm described in the lecture is provided as `medium_find_2d_peak`.

- **(30 points)** Write `quick_find_2d_peak` to compute any peak of array B in $O(n)$ time using the algorithm described in the lecture.