Homework 1 is due March 4 (Monday), 17:30.

Homework submission A pdf copy of your own solutions to Problems 1–3 should be submitted at SUCourse.

Grading Full credit will be given to correct solutions that are described clearly.

Problem 1 (Asymptotic Growth) Rank the following functions by nondecreasing order of growth; that is, find an arrangement $g_1, ..., g_{10}$ of the functions satisfying $g_1 = O(g_2)$, $g_2 = O(g_3)$, ..., $g_9 = O(g_{10})$. All the logs are in base 2.

$$n2^n,\ n\log(n),\ n!,\ n,\ n^{100},\ 2^n,\ \log(n),\ \log(n!),\ (\log(n))!,\ 2^{2^n}.$$

Problem 2 (Recurrences) Give an asymptotic tight bound for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$.

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

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

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

(d) $T(n) = T(n-1) + n$

Problem 3 (Binary Search - Python) Consider the two algorithms for binary search, shown in Figures 1 and 2.[1] Each algorithm takes a sorted list of numbers, `alist`, and a number, `item`, and returns true if and only if `item` is an element of `alist`. The algorithm shown in Figure 1 is iterative whereas the algorithm shown in Figure 2 is recursive.

[1]These algorithms are taken from the book "Problem Solving with Algorithms and Data Structures using Python" by Miller and Ranum: `http://interactivepython.org/runestone/static/pythonds/index.html`.

```
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
```

Figure 1: Iterative binary search

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

Figure 2: Recursive binary search

In the following, n is `len(alist)`.

(a) According to the cost model of Python, the cost of computing the length of a list of size n using the function `len` is $O(1)$, and the cost of extracting a slice of a list using `:` (e.g., `alist[0:n/2]`) is $O(n)$. Based on this cost model:

(i) What is the asymptotic running time of the iterative binary search algorithm shown in Figure 1? (State the running time and give an upper bound for it.)

(ii) What is the asymptotic running time of the recursive binary search algorithm shown in Figure 2? (State a recurrence and solve it.)

(b) Implement these two algorithms using Python. For each algorithm, determine its scalability experimentally by running it with different sized sorted lists of numbers in the worst case. For instance, you can generate a sorted list of n numbers ranging between 1 and 10^7 (using `range` function of Python) and search for 1.

(i) Fill in following table with the running times in seconds.

Algorithm	$n = 10^4$	$n = 10^5$	$n = 10^6$	$n = 10^7$
Iterative				
Recursive				

Specify the properties of your machine (e.g., CPU, RAM, OS) where you run your programs.

(ii) Plot these experimental results in a graph.

(iii) Discuss the scalability of the algorithms with respect to these experimental results.

(iv) Determine whether the experimental results confirm the theoretical results you found in (a).

(c) For each algorithm, determine its average running time experimentally by running it with randomly generated sorted lists of n numbers, and randomly generated keys. For instance, for each randomly generated list of size n, you can generate 50 different keys.

(i) Fill in following table with the average running times in seconds (μ), and the standard deviation (σ).

Algorithm	$n = 10^4$		$n = 10^5$		$n = 10^6$		$n = 10^7$	
	μ	σ	μ	σ	μ	σ	μ	σ
Iterative								
Recursive								

(ii) Plot these experimental results in a graph.

(iii) Discuss how the average running times observed in your experiments grow, compared to the worst case running times observed in (b).

(d) How can we improve the recursive binary search algorithm so that it has the same asymptotic running time as the iterative one? (Make sure that the algorithm is still recursive after the improvements.)

Do we observe this improvement experimentally? (Redo the experiments in (b) for the worst-case analysis, with the improved binary search algorithm. Plot the running times of the given iterative algorithm, the given recursive algorithm, and the improved recursive algorithm as a graph.)