# Exam 1 sample solutions

1. (a) The inner loop is just $\sum_{j=1}^{i} c = ci$ steps. The outer loop runs through $i = n, 3n/4, 9n/16, ..., 1$, so the total number of steps is

$$cn + c\frac{3n}{4} + c\frac{9n}{16} + \cdots + c = \sum_{k=0}^{M} cn\left(\frac{3}{4}\right)^k \leq \sum_{k=0}^{\infty} cn\left(\frac{3}{4}\right)^k = 4cn$$

which is clearly $O(n)$, where $M$ is the number of iterations. (We could calculate $M$, where $n\left(\frac{3}{4}\right)^M = 1$ means that $M = \frac{\log n}{\log \frac{4}{3}} = c_1 \log n$ for some $c_1 > 0$, but we don't actually care since it's a convergent geometric sum. See solution #2 to problem 5 to see how it looks if you don't notice the sum is convergent.)

(b) Inner loop is

$$\sum_{k=1}^{ij} c = ijc$$

steps. The $j$ loop is then

$$\sum_{j=1}^{n} ijc = ic\sum_{j=1}^{n} j = ic\frac{n(n+1)}{2}$$

steps. Finally the outer loop is

$$\sum_{i=1}^{n} ic\frac{n(n+1)}{2} = c\frac{n(n+1)}{2}\sum_{i=1}^{n} i = c\frac{n(n+1)}{2}\cdot\frac{n(n+1)}{2} = \frac{c}{4}(n^4 + 2n^3 + n^2)$$

which is $O(n^4)$.

2. (a) Choose $c = 6$ and $N = 10$. Then for any $n \geq N$, $n^2 > 60$ and so $n^2 - 60$ is positive, so

$$5n^2 < 5n^2 + (n^2 - 60) = 6n^2 - 60 = 6(n^2 - 10)$$

Thus there exist constants $c$ and $N$ such that $5n^2$ is less than $c(n^2 - 10)$ whenever $n \geq N$. We conclude that $5n^2$ is $O(n^2 - 10)$.

(b) Assume that $f \in O(g)$. This means there are constants $c_1$ and $N_1$ such that $f(n) \leq c_1 g(n)$ for all $n \geq N_1$. Choose $c = c_1 + 1$ and $N = N_1$. Then for any $n \geq N$,

$$f(n) + g(n) \leq c_1 g(n) + g(n) = (c_1 + 1)g(n) = cg(n).$$

Thus there exist constants $c$ and $N$ such that $f(n) + g(n) \leq cg(n)$ for all $n \geq N$. Therefore $(f + g)$ is $O(n)$.

3. 
```
Let h be a hashtable where keys are strings, values are integers.
for each s in A
  if h.contains(s)
    h.put(s, h.get(s) + 1)
  else
    h.put(s, 1)
Create array B with the same length as A
i = 0
for each s in A
  B[i] = h.get(s)
  i = i + 1
```

   We are assuming that all hashtable operations are constant time, so there are two consecutive iterations over $A$, each performing constant work per element. Therefore the algorithm is $O(n)$.

4. There are two subproblems whose size is bounded by $n/2$, and a constant amount of work to find the indices for the left and right halves of the array. The base case, where $n = 0$, is also constant time. Therefore an appropriate recurrence is

$$T(n) \leq 2T(n/2) + c, T(0) \leq c$$

5. *Solution 1:* Unrolling as a tree:

   First level has $cn$ steps
   Second level $3c\frac{n}{4}$ steps
   Third level $9c\frac{n}{16}$ steps

   and so on. Total number of steps is

$$\sum_{j=0}^{M-1} cn \left(\frac{3}{4}\right)^j$$

   where $M$ is the height of the tree. Since the sum is a convergent geometric series we can just write

$$cn \sum_{j=0}^{M-1} \left(\frac{3}{4}\right)^j \leq cn \sum_{j=0}^{\infty} \left(\frac{3}{4}\right)^j = 4cn$$

   Hence $T(n)$ is $O(n)$.

   *Solution 2:* if you don't notice that the sum is convergent, it looks like,

$$cn \sum_{j=0}^{M-1} \left(\frac{3}{4}\right)^j = cn \frac{\left(\frac{3}{4}\right)^M - 1}{\frac{3}{4} - 1} = cn \frac{\left(\frac{3}{4}\right)^M - 1}{-\frac{1}{4}} = -4cn \left(\left(\frac{3}{4}\right)^M - 1\right) = 4cn - 4cn \left(\frac{3}{4}\right)^M \leq 4cn$$

*Solution 3:* You could also unroll in this form,

$$\begin{aligned} T(n) &\leq 3T\left(\frac{n}{4}\right) + cn \\ &\leq 3\left[3T\left(\frac{n}{16}\right) + c\frac{n}{4}\right] + cn \\ &\leq 3\left[3\left[3T\left(\frac{n}{64}\right) + c\frac{n}{16}\right] + c\frac{n}{4}\right] + cn \\ &= 3^3T\left(\frac{n}{64}\right) + 3^2c\frac{n}{4^2} + c\frac{n}{4} + cn \\ &\leq 3^MT(2) + \sum_{j=0}^{M-1} cn\left(\frac{3}{4}\right)^j \end{aligned}$$

The sum on the right is $O(n)$ as above. From $n\left(\frac{1}{4}\right)^M = 2$ we get $M = \frac{\log n - 1}{2}$ (here $\log n$ denotes $\log_2 n$). Then $3^MT(2) = c \cdot 3^M = c \cdot 3^{\frac{\log n-1}{2}} = \left(\sqrt{3}\right)^{\log n - 1} = n^{\log\sqrt{3}} \cdot \frac{1}{\sqrt{3}}$, which is also $O(n)$ since $\log\sqrt{3} < 1$. Thus $T(n) \in O(n)$.

6. *Solution 1:*

```
FindMaxDiff(A):
  if A has only two elements
    return A[1] - A[0]
  L = left half of A
  R = right half of A
  maxdiff_L = FindMaxDiff(L)
  maxdiff_R = FindMaxDiff(R)
  max_R = largest element of R
  min_L = smallest element of L
  return the maximum of maxdiff_L, maxdiff_R, (max_R - min_L)
```

There are two subproblems of size $n/2$. Finding the max of the right half and the minimum of the left half are $O(n)$ operations, as is splitting the array into $L$ and $R$. Therefore the recurrence is

$$T(n) \leq 2T(n/2) + cn, T(2) \leq c.$$

*Solution 2:* We can do a little better by noticing that there is some wasted effort to find max of $R$ and min of $L$ each time we return from a recursive call. Let's rewrite the algorithm so that in addition to returning the maximum difference from each half of the array, it also returns the minimum and maximum values. We can also eliminate the potential $O(n)$ work to create the subarrays $L$ and $R$ by using indices instead.

```
FindMaxDiff(A):
  let (maxdiff, min, max) = FindMaxDiffRec(A, 0, n - 1)
  return maxdiff

FindMaxDiffRec(A, left, right)
  if right - left = 1
    min = the smallest of (A[left], A[right])
    max = the largest of (A[left], A[right])
    return (A[right] - A[left], min, max)
  mid = (left + right) / 2
  (maxdiff_L, min_L, max_L) = FindMaxDiffRec(L, left, mid - 1)
  (maxdiff_R, min_R, max_R) = FindMaxDiffRec(R, mid, right)
  maxdiff =  maximum of maxdiff_L, maxdiff_R, (max_R - min_L)
  max = larger of max_L and max_R
  min = smaller of min_L and min_R
  return (maxdiff, min, max)
```

Now the work to merge the subproblem solutions into an overall solution is constant, since to find the min and max we just have to compare the min and max values returned from the recursive calls. A recurrence for the time complexity $T$ is

$$T(n) \leq 2T(n/2) + c, T(2) \leq c.$$