Some sample problems for Exam 1
Exam date: Thursday, October 3, 8:15 - 9:45 pm
Sections A and D: Kildee 0125
Section C: Gerdin 1148 (NOTE CHANGE IN LOCATION)

- In general you may use the following algorithms and their time bounds as a "black box" on the exam (that is, you do not need to write the code nor derive runtime for them): *sorting algorithms, merging two sorted arrays, extracting the min (max) from a min-heap (max-heap), heap add, remove, percolate up, percolate down, heap construction, BST inorder traversal, counting inversions, efficient big integer multiplication, closest pair of points in a plane.* You may assume that adding, finding, or removing a hashtable element is O(1) except when hashing variable-sized objects (such as strings). *However, if you modify any of these methods, then you must write a complete description of the modified method. Merely stating the modification does not suffice.*
- For any problem or sub-problem, if you write "*Do not grade*" and nothing else, you will receive 20% credit for it.
- For all algorithm problems, part of the grade depends on the efficiency.
- The following information will be included on the exam:

  $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

  $\sum_{i=0}^{M} r^i = \frac{r^{M+1}-1}{r-1}$, $\sum_{i=0}^{M} 2^i = 2^{M+1} - 1$

  $\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$ when $0 < r < 1$

  $b^{\log_b x} = x$, $\log_b x^y = y \log_b x$, $\log_b xy = \log_b x + \log_b y$, $\log_b \frac{x}{y} = \log_b x - \log_b y$

  $\log_a x = \frac{1}{\log_b a} \log_b x$, $x^{\log_b y} = y^{\log_b x}$

  For every $k > 0$ and $a > 0$, $n^k \in O(n^{k+a})$, and $\log^k n \in O(n^a)$.
- If you want more practice problems, it is usually possible to find lots of examples by googling, e.g., "interview questions divide and conquer" or something like that.

1. Prove or give a concrete counterexample for each of the following:

   (a) $5n^2$ is $O(n^2 - n)$

   (b) 5^n is $O(n!)$

(c) $4n^2 - 3n^2(1 - \log_2 n)$ is $O(n^2)$

(d) if f is $O(h)$ and g is $O(h)$ then fg is $O(h)$, where fg is the function defined by $fg(n) = f(n) \cdot g(n)$.

(e) if $f(n)$ is $O(\sqrt{(g(n))})$, then $2^{f(n)}$ is $O(2^{(g(n))})$

2. Derive a (tight) Big-O runtime bound. (You must show your work.)

```
m = 1
for i in the range [1, n]
   for j in the range [1, m]
     Constant Number of Operations
   m *= 5
```

3. Derive the runtime:

```
for i in the range [1, n - 1]
   min = i
   for j in the range [i , n]
     if a[j] < a[min]
       min = j
   swap a[i] with a[min]
```

4. Derive the runtime:

```
k = n
while k > 1
   for j in the range [1, 99n]
     Constant Number of Operations
   k *= .99
```

5. Suppose that the cost to multiply a k-digit number with an m-digit number is $O(mk)$. (Note that in general the product would have up to $m + k$ digits.) Consider the following simple algorithm for finding x^n for an integer power n:

```
pow(x, n):
  p = 1
  for i in the range [1, n]
    p = p * x
  return p
```

If k denotes the number of digits of x, what is the complexity of this algorithm (as a function of n and k)?

6. The following algorithm finds the successor (node containing the next largest key) of a given node in a binary search tree, returning null if there isn't one.

```
successor(node):
  if node has a right child
      // find leftmost entry in right subtree
      current = node.right
      while current has a left child
          current = current.left
      return current
  else
      // go up the tree to the closest ancestor that is
      // a left child; its parent must be the successor
      current = node.parent;
      kid = node;
      while current is not null and kid is current's right child
          kid = current
          current = current.parent
      // either current is null, or kid is left child of current
      return current;
```

Then we can create an array of the elements of the tree in ascending order as follows:

```
i = 0
current = root
while current has a left child    // find smallest element
  current = current.left
while current is not null
  a[i++] = current.key
  current = successor(current)
```

What is the complexity of the iteration strategy above?

7. Let S be a set of strings of length k, and let $n = |S|$. Let h be a hash function that maps strings of length k to $\{0, 1, \cdots, m-1\}$. Assume that the time taken to compute $h(x)$ is $O(k)$. Suppose that H is a hashtable (with m buckets) constructed to store S by using the hash function h. Let M be the maximum load of the hash table H. Recall that if z is any string of length k, the time taken to search whether z appears in the hashtable is $O(k + Mk)$, since it is $O(k)$ to compute $h(z)$ and $O(Mk)$ to search for z in the list at $H[h(z)]$ (since determining whether two k-length strings are equal is $O(k)$). Similarly, the time taken to delete z is $O(k + Mk)$.

   Let g be another function from k-length strings to 32-bit integers such that computing $g(x)$ is $O(k)$, and such that g is one-to-one on S, i.e, for every x and y from S,

   $$x \neq y \Rightarrow g(x) \neq g(y)$$

   Can you design a new hash table of size m, that uses g in conjunction with h, such that the time taken to search/delete is $O(k + M)$? (You must justify your answer, i.e., explain the design or explain why it can't be done.)

8. Let a and b denote two integer arrays of length n. Write an $O(n)$ algorithm to determine whether a is a permutation of the elements in b (i.e., same values in a different order). There may be duplicates.

9. Write an efficient algorithm to find the maximum element in a min-heap (implemented as an array-based binary heap). Determine the runtime of your algorithm.

10. In Com S 227 you may have seen the following code for printing out a solution to the Towers of Hanoi puzzle for n discs:

```
public static void move(int n, String srcPeg, String dstPeg, String extraPeg)
{
  if (n == 1)
    System.out.println("move disc from " + srcPeg + " to " + dstPeg);
  else
  {
    move(n - 1, srcPeg, extraPeg, dstPeg);
    move(1, srcPeg, dstPeg, extraPeg);
    move(n - 1, extraPeg, dstPeg, srcPeg);
  }
}
```

    Let $T(n)$ denote the runtime of this algorithm. Write a recurrence for T and solve it.

11. Derive a solution for each of the following:

    (a) $T(n) \leq 3T(\frac{n}{3}) + cn$, $T(2) \leq c$

    (b) $T(n) \leq T(\frac{n}{4}) + T(\frac{3n}{4}) + cn$, $T(2) \leq c$

    (c) $T(n) \leq n^{\frac{1}{2}} T(n^{\frac{1}{2}}) + cn$, $T(2) \leq c$ (*Tip:* first try unrolling it for $n = 256$.)

    (d) $T(n) \leq 2T(\frac{n}{2}) + c\sqrt{n}$, $T(2) \leq c$

12. It is possible to compute integer powers x^n using a divide and conquer style algorithm based on the relations $x^n = x^{\frac{n}{2}}$ (n even) and $x^n = x \cdot x^{\frac{n-1}{2}}$ (n odd).

    (a) Write the algorithm.

    (b) Write a recurrence for the runtime T, assuming that cost to multiply a k-digit number with an m-digit number is $O(mk)$. Assume that if x has k digits, then x^n will have nk digits, and that p^2 has twice as many digits as p. For your recurrence, it will be ok to assume that n is always even.

    (c) Solve your recurrence, as a function of n and k, where k is the number of digits of the input x.

13. Write a divide and conquer algorithm to find the greatest common factor of list of integers. For example, given [105, 42, 98, 14], the greatest common factor is 7. Assuming that you can find the greatest common factor of *two* integers in constant time, write a recurrence for the runtime of your algorithm, and solve it.

14. Write a divide and conquer algorithm to find the maximum sum of any subarray of a given array of integers. For example, in the array [10, 2, -8, -3, -1, 3, -1, 7, 4, -5], the maximum sum is 14 (seen in subarray [3, -1, 7, 4]). Write a recurrence for the runtime of your algorithm, and solve it. (*Tip:* If you choose a fixed index i, how hard is it to find the maximum sum of the subarrays that include index i?)