CS180 – Summer 2012 **Name**(last, first): _____

July, 16th, 2012 **UID**: _ _ _ - _ _ - _ _ _ _

Midterm Exam

Total Time: 120 minutes

- **Do not** open this exam booklet until you are directed to do so. Write down your name and UID or points will be taken off.
- This exam is **closed book, closed notes.**
- No electronic equipment allowed (cell phones, computers, etc.)
- Read the questions carefully.
- Write legibly. What cannot be read will not be graded.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Do not put part of the answer to one problem on the back of the sheet of another problem.
- Plan your time wisely. Do not spend too much time on any one problem. Read through all of them first and attack them in the order that allows you to make the most progress.
- Show your work, as partial credit will be given. You will be graded on correctness as well as on clarity, so be clear in explaining your work.
- If a problem asks you to write pseudocode, the conventions discussed in class must be followed very closely.
- Always remember to analyze the time complexity of your solution.
- If you have a question about the meaning of a question, raise your hand.

Good luck!

Problem	Points	Score
1	2	
2	7	
3	7	
4	9	
5	7	
6	8	
7	7	
8	3	
Total	50	

CS180 – Summer 2012 **Name**(last, first): _____

July, 16th, 2012 **UID**: _ _ _ - _ _ - _ _ _ _

Problem 1 (2 points)
Write down the recurrence for the merge sort algorithm and provide its running time using O-notation. Do not write any pseudocode. You are not required to show your work.

Merge sort recurrence: $T(n) = 2T(n/2) + O(n)$
$T(n) = O(n\log n)$

Problem 2 (7 points)
Suppose you are choosing between the following three divide-and-conquer algorithms:

- Algorithm A solves problems by dividing them into eight subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size (n – 1) and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of one third the size, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms, and which would you choose? Show your work and justify your choice.

1) Recurrence: $T(n) = 8T(n/2) + O(n)$
   $f(n) = O(n^{\log_b a - \varepsilon})$?
   $n = O(n^{\log_2 8 - \varepsilon}) = O(n^{3-\varepsilon}) \Rightarrow \varepsilon = 1$
   By case 1 of the master method, the running time is: $T(n) = \Theta(n^3)$

2) Recurrence: $T(n) = 2T(n-1) + O(1)$
   Solution by iterative substitution (same as on Quiz, problem 4 (1))
   $T(n) = O(2^n)$

3) Recurrence: $T(n) = 9T(n/3) + O(n^2)$
   By case 2 of the master method, the running time is:
   $T(n) = \Theta(n^2 \log n)$

I would choose Algorithm C because it is asymptotically faster than both A and B.
For n = 100, Algorithm A takes 1,000,000.
Algorithm B takes 1,267,650,600,228,229,401,496,703,205,376
Algorithm C takes 66,438.56

CS180 – Summer 2012 **Name**(last, first): _____

July, 16th, 2012 **UID**: _ _ _ - _ _ - _ _ _ _

Problem 3 (7 points)

A polygon is convex if all of its internal angles are less than 180° (and none of the edges cross each other). The figure below shows an example. We represent a convex polygon as an array V[1…n] where each element of the array represents a vertex of the polygon in the form of a coordinate pair (x,y). We are told that V[1] is the vertex with the minimum x coordinate and that vertices V[1…n] are ordered counterclockwise, as in the figure. You may also assume that the x coordinates of the vertices are all distinct, as are the y coordinates of the vertices.

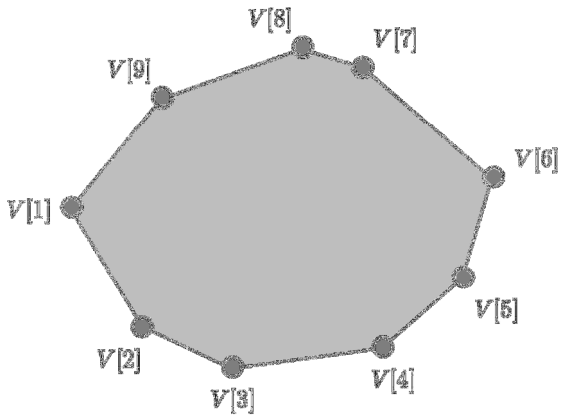


Figure 1: An example of a convex polygon represented by the array V[1…9]. V[1] is the vertex with the minimum x-coordinate, and V[1…9] are ordered counterclockwise.

a) Give an algorithm to find the vertex with the maximum x-coordinate in O(log n) time. Provide pseudocode and follow the pseudocode conventions.

Consider the problem only in the x dimension. Let the x-coordinates of the vertices V[1..n] be X[1..n]. Note that X increases to a maximum value somewhere in the array, then decreases back to X[1]. The maximum x value can be solved as a Unimodal Search problem.

Unimodal search works by checking adjacent elements in the array. If the element X[i] is smaller than the element after it, X[i+1], then the max lies further down the array, ie in X[i+1..n]. Similarly, if the element X[i] is larger than the element after it, X[i+1], then the max lies before it in the array, ie in X[1..i]. This divide and conquer method is similar to binary search so the running time is O(log n).

Pseudocode:

Algorithm UnimodalSearch(X):

Input: X is array that has elements that increase then decrease

Output: the maximum value in X

```
a ← 1
b ← n
while (a < b)
   mid ←  (a+b)/2
   if X[mid] < X[mid+1] // array is increasing, so max occurs after
      a ←  mid + 1
   if X[mid] > X[mid+1] // array is decreasing, so max occurs before
      b ←  mid
return X[a]
```

b) Give an algorithm to find the vertex with the maximum y-coordinate in O(log n) time. Provide pseudocode and follow the pseudocode conventions.

After finding the element with the maximum x value, given as V[max], we see that the y values of the vertices V[max], V[max+1], ... , V[n-1],V[n],V[1] form a unimodal array. The same algorithm can be applied now in the y dimension with this subset of vertices. The running time is again O(log n).

CS180 – Summer 2012 **Name**(last, first): _____

July, 16th, 2012 **UID**: _ _ _ - _ _ - _ _ _ _

Problem 4 (9 points)

a) What is the difference between a max-heap and a binary search tree?

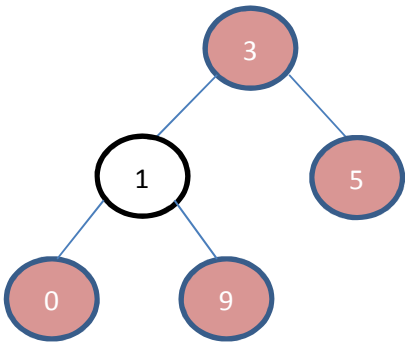
A heap is a binary tree with 2 additional properties:

1) Shape property: complete binary tree with a possibly incomplete last row that is filled from left to right
2) Heap property: the value stored in each node has to be greater than or equal to the values stored in both of its children.

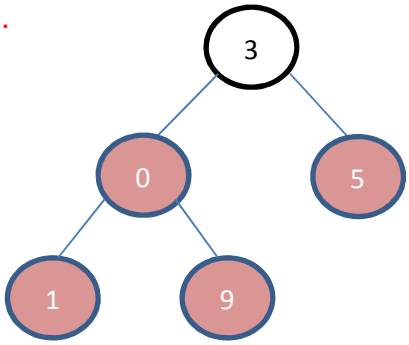
b) Show all the steps required to transform this array into a min-heap. Draw each step out and write a sentence explaining each of the steps.

3	1	5	0	9

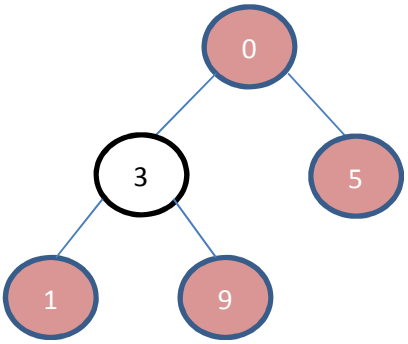
Build a tree out of the array by using the indexes to number the nodes on each level



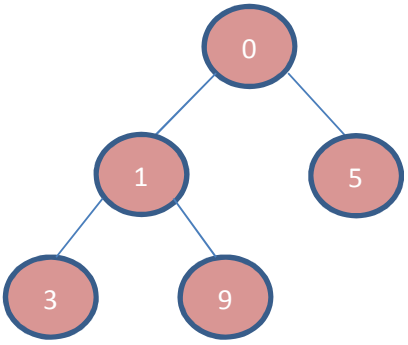
Start at the first non-leaf (internal) node and check for the heap property. It is violated so swap with smaller left child. Since the swapped child is a leaf, we don't have to check for the heap property down its subtree.



The next internal node is the root node. Check for the heap property. It is violated so swap the root with the smaller left child.

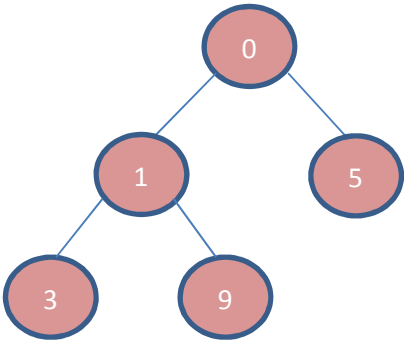


Now the heap property is violated on the left subtree, so swap the root of the subtree with the smaller left child to produce a min-heap.

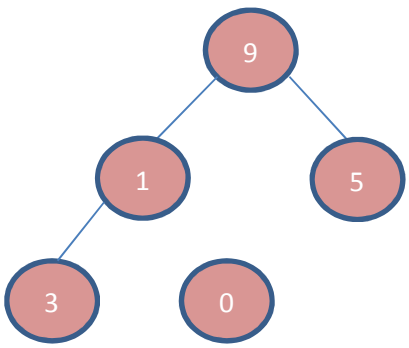


c) Use the min-heap from part b) to sort the array in descending order. Draw each step out and write a sentence explaining each of the steps.

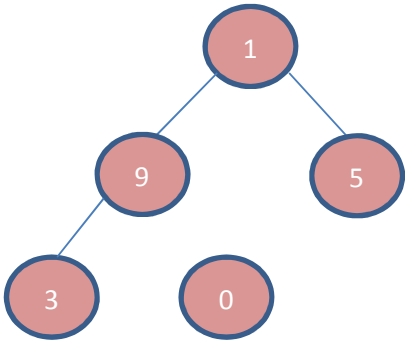
Heap from part b)

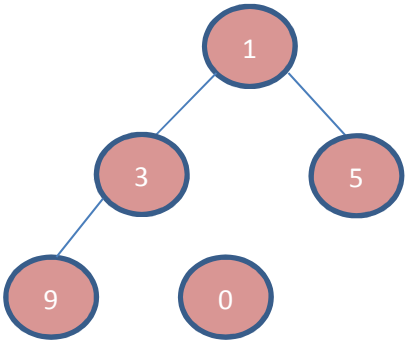


Swap the min node (root node in a min-heap) with the last node.

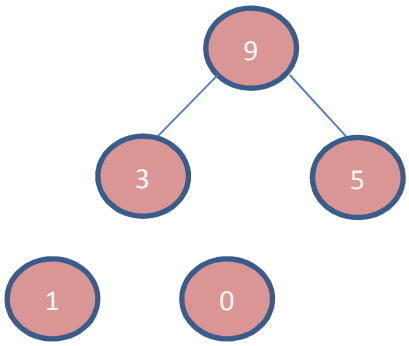


After the swap, the heap-property is checked and the tree heapifyed, if the property is violated.

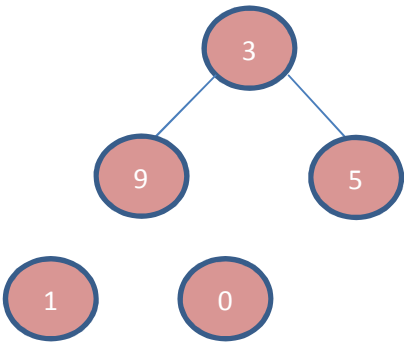




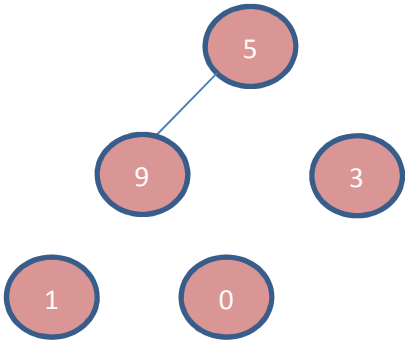
Now we extract the min again and swap it with the last element.



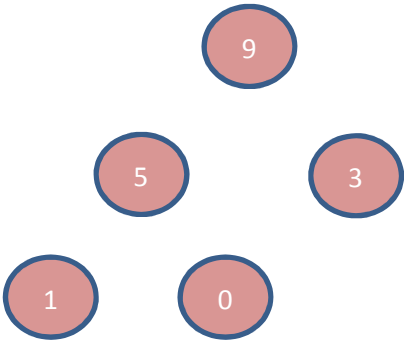
After the swap, the heap-property is checked and the tree heapifyed if the property is violated.



Swap the min with the last element. No heapifying necessary.



Swap the min with the last element. The numbers are now sorted in descending order.



Problem 5 (7 points)

Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

a) One strategy is to use the merge procedure used in the merge sort algorithm to merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of k and n? Explain.

Same as homework 2, problem 5a.

b) Give a more efficient solution to this problem, using a heap data structure. Give a description of your algorithm in English and analyze its time complexity.
Use the following format to describe your algorithm:
Step 1: XXXXXX;
Step 2: XXXXXX;
Step 3: XXXXXX, etc.
Clarify any notation you use in the description.

Step 1: Given k sorted arrays, take each of the smallest elements of the k arrays and build a min-heap of k elements. This step takes time O(k) or O(klogk).

Step 2: Take the minimum of the heap, which is the smallest element of the whole k arrays. Time O(1).

Step 3: Replace this minimum element of the heap with the next smallest element in the same array that the minimum came from. Run min-heapify on this new heap. Time O(log k).

Repeat steps 2&3 until all n x k elements have been processed. Since steps 2&3 are repeated nk times and each time takes O(log k), the running time of the algorithm is O(nk log k).

Note that this is the same time as the divide and conquer approach as in the homework. To implement this, we need to be able to track which array each element originated from, which can be done by extending the heap data structure.

Problem 6 (8 points)

Consider a sequence $(x_1, x_2, \ldots, x_n)$ of n unsorted nonnegative integers, where n is even and is in the range 0 to 100. Design an O(n)-time algorithm that partitions the numbers into n/2 pairs. The sum of the numbers in the pairs are denoted by $(S_1, \ldots, S_{n/2})$. Your algorithm should minimize the maximum sum.

Example: if $(x_1, x_2, \ldots, x_n) = (4, 9, 3, 2)$ then a possible pairing is (4,2) and (3,9). The resulting sums are $S_1 = 4 + 2 = 6$ and $S_2 = 3 + 9 = 12$. The maximum sum of these two pairs is $S_2 = 12$. The pairing (3, 4) and (2, 9), however, produces a smaller maximum sum: 2 + 9 = 11 and should, therefore, be the output of your algorithm.

Provide the pseudocode for this algorithm and analyze its time complexity. If you use a well-known algorithm, also provide the pseudocode for it.

With a few test cases, it should be clear that the solution to this problem is the pairing of (smallest, largest), (2nd smallest, 2nd largest), (3rd smallest, 3rd largest), and so on. (You are not required to prove this result).

Given a sorted array, we can step through the first half array and match each element with its corresponding element from the other half to form the pairs. Time is O(n).

To sort an unsorted array in linear time, we can use counting sort as long as the conditions are satisfied, which they are in this problem (range of elements is constant k).

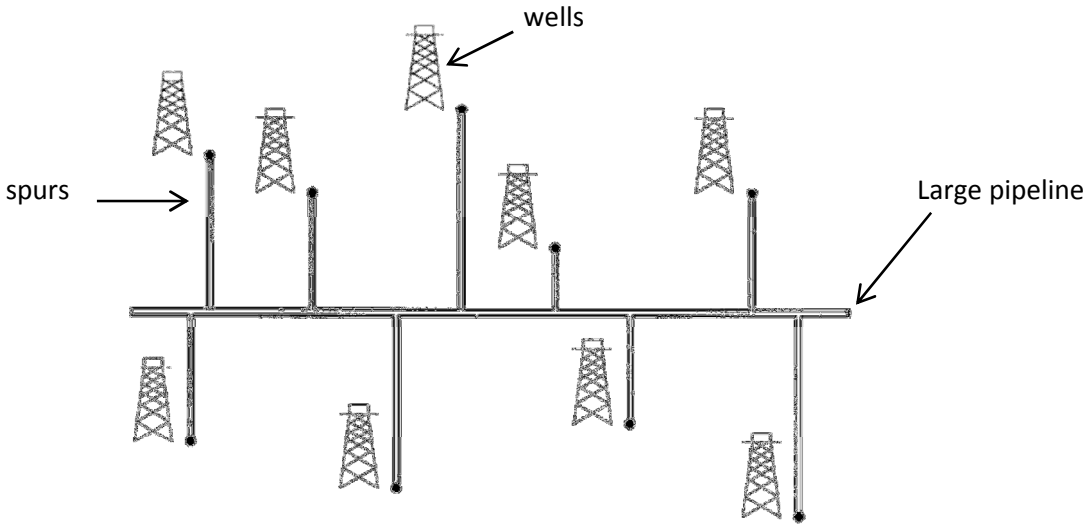
Pseudocode:

```
Algorithm CountingSort(A):
   Input: A is unsorted array
   Output: sorted array

..... Countingsort given in textbook and in lecture 5 notes .....

Algorithm MakePairs(A):
   Input: A sorted array A[0..n-1]
   Output: B array [A[1st],A[last],A[2nd],A[2nd last], ... ]

   for i ← 0 to floor(n/2) - 1 do
     B[2i] ← A[i]
     B[2i+1] ← A[n-i]
  return B
```

Problem 7 (7 points)

a) Professor Olay is consulting for an oil company, which is planning a large pipeline running east to west through an oil field of n wells. The company wants to connect a spur pipeline from each well directly to the main pipeline along the shortest route (either north or south), as shown in the figure. Given the x- and y-coordinates of the wells, how should the professor pick the optimal location of the main pipeline, which would be the one that minimizes the total length of the spurs? Show how to determine the optimal location in linear time. Give a description of your algorithm in English. If you choose to use a well-known algorithm, provide the steps of that algorithm.

Use the following format to describe your algorithm:
Step 1: XXXXXX;
Step 2: XXXXXX;
Step 3: XXXXXX, etc.
Clarify any notation you use in the description.



Since the pipeline runs east-west and the spurs north-south, then we do not need to consider the x dimension. The problem reduces to finding the optimal point in the y dimension. If you pair up the wells on either side of the pipeline, then the position of the pipeline doesn't affect the total length of the spurs. If n is odd, however, then the pipeline has to pass through the median wellto minimize the total length of the spurs. To find the median well we use the 5 steps of the selection algorithm with k = n/2 (wells are ordered on the y-axis).
1. split the wells into groups of 5 and a leftover group if n is not divisible by 5
2. sort the wells' y-locations using selection sort and pick the median well of each group
3. recursively apply the 1st and 2nd steps until we find the median of medians
4. use the median of medians as a pivot and partition the wells (wells with a smaller y-location to the left of the pivot, and wells with larger y-locations to the right of it).

5. recursively apply the algorithm (5 steps) to the appropriate partition. If k = the rank of the pivot in the array, we found the median. Otherwise if k < the rank of the pivot, recurse on the left partition. If k > the rank of the pivot, recurse on the right partition.

Problem 8 (3 points)

Assume you have an array A with n d-digit elements. The range of the n numbers is [0;k]. If you want to use the radix sort algorithm to sort the array, which of the following 2 algorithms would you use?

a) Bubble sort
b) Heapsort

Justify your answer and provide the time complexity for the new radix sort algorithm.

Radix sort requires a stable algorithm for sorting the digits in order to function correctly. Out of the two algorithms, bubble sort is the stable one and is therefore the correct one use. The time complexity of the new radix sort implementation is $O(dn^2)$.