Liang Tao
CS323 Data Structures and Algorithm
February 2, 2014

Assignment 1

Problem 1

1a) To solve the problem, 1) first we have to build a heap, and then 2) repeatedly remove the M max numbers in the heap.

1) For the first step, read all the N numbers into an array which has the size of N. Then we will process the sink( ) making sub-heaps which is described in the book. Since every position in the array is the root of a small sub-heap, so we can first use sink( ) to make such sub-heap, then calling it on that node makes the subtree rooted at the parent a heap. So eventually this process can establish the heap inductively.

2) For the second step, we just need a loop to remove M max numbers in the ordered heap and print them out.

Analysis:

As the book described, sink-based heap constructions cost less than 2N compares and less than N exchanges to construct an ordered heap for N inputs. Because all the construction steps are based on the N-size array, the space we used is O(N).

For the runtime, because in an N-key heap, a "remove maximum" step cost no more than 2lgN compares, so M compares would cost no more than M*(2lgN) compares. Since M<= N/lgN, total runtime cost would be 2N + 2M*lgN <= 4N ~ O(N). Then we have fulfilled the requirement of the problem.

1b) For this problem, we have to 1) first build a heap, and 2) traverse all the N numbers, repeatedly insert the each number in the heap, and after insertion, deleting minimum once the size of the heap reach M.

1) For the first step, what we need to do is building an array which has a size of M.
2) Then as stated above, we use a for loop to insert every number into the array-based heap, and delete minimum after each insertion once the size of the heap reach M.

Analysis:

The heap is based on the array, and all the following steps do not need extra space to process, so space we require is O(M);

Since based on the proposition, heap algorithm requires no more than 2lgN compares for remove the maximum in an N-key heap, then our algorithm only need N*2lgM = 2NlgM ~ O(NlgM) time.

Then we have fulfilled the requirement of the problem.

Problem 2

1a) **Claim**: " A tree of height h constructed by the weighted quick-union algorithm has at least 2^h nodes. (h is the height of this tree)"

Proof:

Base case: (h = 0): show P (0).

If this kind of tree has a height of 0, then the tree is just an isolated root node with no children. So its size is 1, then $2^0 = 1$ is true.

Inductive step: Supposing P(h), show P(h+1).

Supposing T is such a tree of height h+1, then we need to argue that size(T) is at most $2^{(h+1)}$.

To create a T, we linked a tree T1 as a new child of the root of tree T2, where T1 has a height of h. By inductive hypothesis, T1 had nodes at least 2^h , and by the weighted linking rule, T2 has at least the size of T1, then T2 has at least the size of 2^h.

So T has size(T) = size(T1) + size(T2) >= $2^h + 2^h = 2^{(h+1)}$.

So by induction, P(h) is true for all h. So the claim is true.

2b) **Claim**: "In a tree constructed by the weighted quick-union algorithm he number of nodes at distance at most k steps from the root is at least (h, <=k). (h is the height of this tree)"

Proof:

Base case: (h = 0): show P (0).

If this kind of tree has a height of 0, then the tree is just an isolated root node with no children. So k must be 0 or 1, then P(0) is absolutely true.

Inductive step: Supposing P(h), show P(h+1).

Supposing T is such a tree of height h+1, then we need to argue that the number of nodes at distance at most k steps from the root is at least (h+1, <=k).

To create a T, we linked a tree T1 as a new child of the root of tree T2, where T1 has a height of h. By inductive hypothesis, T1 has least (h, <=k) nodes at distance of k from root, which means nodes number >=C(h,k) + C(h, k-1) + C(h, k-2)+……C(n,0) is true.

To prove (h+1, <=k) is true, we need to prove that (1) C(h+1, k) + C(h + 1, k-1) + C(h +1,k-2) + … + C(h+1,0) is true. Because C(h+1,k) = C(h,k) + C(h,k-1), so the formula 1) would be C(h,k) + C(h,k-1) + C(h, k-1) + C(h,k-2) + C(h,k-2) + ……C(h,1) + C(h,1) + C(h,0), so we just need to prove that 2) C(h,k-1) + C(h,k-2) + C(h,k-3) + C(h,1) is true. Since we link T1 to T2, similarly, the formula 2) would be true under the P(h), the P(h+1) would be true under this circumstance.

So by induction, P(h) is true for all h. So the claim is true if we assume that we can only create a tree of height h+1 by linking two trees of height h.

Problem 3

In this problem, we can first sort the sublists, and then merge them into a single file.

For the first step, we do the following process:

1. Create an array with size P^2, read P^2 numbers in P pages from disk into an array in O(P) time.
2. Sorting the array with heap sort which costs less than $2P^2$ compares and P^2 exchanges. So the total runtime cost of heap sort would be $(2P^2\log P^{2} + 2P^2)$ $\sim O(P^2\log P^2) = O(P^2\log P)$, and $O(\log P^2) = O(\log P)$.
3. At last we will write the sorted numbers back into the disk.

Analysis

Runtime = $O(P^2\log P)*P = O(P^3\log P)$.

Space = $O(P^2)$.

For the second merge part,we do the following process:

1. Suppose we have P input streams and an output stream, and each steam has P words. For the input stream, each of them corresponds to a sublist of the sorted file above.
2. Create a IndexMixPQ with the size P to merge data and keep track the minimum index. Each index corresponds to an input stream.
3. Start to read the input streams.
   While sublist still have unread pages
   - if an input stream is empty, read next page from sublist
     - do loop
       - a. read and insert next number from every input stream into IndexMinPQ client.
       - b. Find the smallest number in the IndexMinPQ, removes the corresponding entry, then adds a new entry for the next number in that stream

Analysis

Runtime: delMin( ) costs $P^3\log P$, same as insert( ), so the total cost would be $O(P^3\log P)$.

Space: $P^2 + P \sim O(P^2)$, P^2 for the input streams and P for the output stream.