Ph D Questions for 503 Half of May 2014 Exam

Please answer all questions.

Question I. This question concerns, the subroutine Heapify, the algorithm Heapsort and its BuildHeap subroutine. By a heap, I mean a "Max-Heap" where each parent stores a key at least as large as its children. The goal of heap-sort will be to produce a sorted listed in positive order (e.g. lowest keys occur first and largest keys occur last). Some parts of this question will concern hand-drawn pictures of heaps, that will be handed out to the class. Please answer all parts of this question.

1. The picture in handout Diagram (a) is actually not a heap. However, it will be transformed into a heap, if you call the procedure "Heapify" with its data structure as an input. Each operation of "Heapify" will swap the contents of two nodes, called X and Y. Please list for me all the swap operations, produced by Heapify, in the chronological order they are done. Your answer should therefore look something like

   Swap(P,Q) followed by Swap(S,T) followed by Swap(U,V)

   Then when you are done doing the above, draw for me the resulting heap.

2. Repeat the procedure above for the pictures (b), (c) and (d) (of stating the swaps and drawing the resulting heaps).

3. For any node i, *please first tell me* the addresses of its left and right child. Next consider the array below that stores 200 in the address 1, 100 in the address 2, 66 in address 3 ... and 42 in address 13. Draw the picture of the heap (tree-like structure) that begins with the address 2 (storing 100) and consists of the subtree descending from this address. (You will avoid careless errors if you give the 13 keys below addresses of 1-13 on a piece of scrap-paper.)

200, 100, 66, 90, 70, 45, 47, 50, 55, 34, 30, 41, 42

4. Assume that "Heapify(A,i)" is an invocation that asks the "Heapify" subroutine to heapify the fragment of array A that lies below the address i. *You do not have to encode this Heapify subroutine.* However, I would like you to write the code of the BuildHeap procedure that takes an array of A of length N and transforms it in O(N log N) time into an array that satisfies the max-heap property (defined in the first paragraph of this question) via repeated employments of the "Heapify" subroutine. Also explain, how Buildheap would process the input array given below. Your answer should consist of 4 drawings because BuildHeap will call Heapify EXACTLY four times. What I want you to do is show how the array below changes after each iteration of "Heapify". Thus, all you need to do this question, is to draw four arrays, indicating first, second, third and fourth changes in array after each Heapify *plus additionally* write the tiny amount of code for mainline fragment of Buildheap.

   50, 25, 10, 70, 40, 22, 5, 90, 80

5. The next and main part of the HeapSort Algorithm unravels a heap to build a fully sorted list in O(N log N) time. In regards to this mainline procedure, please answer A and B:

   A. The mainline of HeapSort merely swaps two elements and calls a special variant of Heapify to subsequently do a correction. Please write down, formally, the code for this main-line procedure. (It is only about half-a-dozen lines long when you are given the permission to avoid writing the code for Heapify's subroutine, as I am providing you.)

   B. If the procedure (above) was given the tiny 6-element heap below, please indicate how it will build a sorted array by repeatedly doing a swap, followed by a Heapify, followed by a Swap, until it is done. What I want you to do is to indicate how this list changes after each Swap-Heapify cycle. (Each picture should be simply a new arrangement of the order in the list below after a Swap-Heapify cycle is done.)

      100, 40, 60, 2, 5, 13

      Thus, there will be several cycles of Swap-Heapify, and you should indicate how the above list changes during each cycle.

6. In 2-to-5 sentences (and definitely not more), please tell me how Heapsort improves upon

the memory space of Merge Sort. (Don't waste much time on this question because it is not worth many points.)

7. Both Heapsort and Merge Sort respect an $\Omega(NlogN)$ lower bound for sorting, but Radix Sort is in some isolated senses faster. Again in no more than say 2-to-5 sentences, tell me why Radix Sort is faster. (Don't waste much time on either this question or Item 6 because neither will be worth more than a tiny amount of points.)

Question II. Issues Involving Hashing

A) Define the Separate Chaining Rule for resolving collisions during hashing (known as the "chained hash method" in your textbook). Then show what a hash file with ten slots would look like when you insert the following sequence of records into a file that uses the separate chaining rule for resolving collisions and which inserts a key K into the address " K mod 10 ".

** 42, 46, 62, 32, 36, 28, 77, 99 (keys OBVIOUSLY inserted in chronological order)

B) Please next provide a 1-sentence definition for the "open addressing" method for resolving hash collisions (which technically involves an infinite number of hash functions H_1, H_2, H_3, ...). Also, define the simplified version of this "open addressing" that uses the "Linear Probing" method for resolving collisions. You actually should provide two simple essentially equivalent 1-sentences of the Linear Probing methods where:

1. One definition of "Linear Probing" views it as a special case of "open addressing" method for resolving hash collisions for a file with S slots.

2. The other definition of "Linear Probing" is the trivial 1-or-2 sentence rewrite of the above definition that does not, *technically,* refer to the "open addressing" method when formally defining the "Linear Probing" method for resolving collisions in a hash file with S slots.

Finally AND MOST IMPORTANT OF ALL, draw a picture of what a file of 10 slots would look like when you insert elements from the sequence of ** in the CHRONOLOGICAL order into a hash file with 10 slots. Once again assume that the rightmost digit of each key is used to

determine its initially chosen hash location. (BE CAREFUL TO DOUBLE CHECK your answer to make certain that no careless errors occur.)

C) Thirdly consider a version of the "Double Hashing" method for resolving collisions, where we insert in *chronological order*, the keys from the sequence ** into a file with s=10 slots under the rules that the rightmost digit of each key determines its initial placement and the key's leftmost digit determines its employed displacement (under double hashing). First again provide two simple and equivalent 1-or-2 sentence definitions of such a Double-Hash method where:

1. The first definition for "Double Hashing" views it as a special case of "open addressing" method for resolving hash collisions for a file with S slots.

2. The second definition of "Double Hashing" is a trivial 1-or-2 sentence rewrite of the above definition that does not, *technically,* refer to the "open addressing" method when defining the "Double Hashing" method for resolving collisions in a hash file with S slots.

Finally AND MOST IMPORTANT OF ALL, draw a picture of what a file of 10 slots would look like when you insert elements from the sequence of ** in the CHRONOLOGICAL order into a hash file with 10 slots under the above Double-Hashing Method. **Once again assume** that the second digit of each key is used to determine its initially chosen has location. (BE CAREFUL TO DOUBLE CHECK your answer to make certain that no careless errors occur.)

D) Next, let us recall that the i-th universal hash for a file with S slots and prime number P maps a key K onto a hash-chosen address " $H_i^{S,P}(\ K\ )$ ", where

1. S is the number of slots in the hash file,

2. P is a prime number whose bit-length is longer than the number of bits in any input key,

3. K is the "key" of the record that is being mapped into a hash address

4. and i designates the hash function' s "index", subject to the constraint that $0 < i < P$.

In such a context, please first write down the precise address " $H_i^{S,P}(\ K\ )$ ", that universal hashing will map the key K onto. Then, assuming that $S = 10$ and $P = 31$, and (i, K) is one of the ordered pairs of (11,20) or (50,5) or (7,11), tell me what address they map onto. (You obviously will not be

able to answer the second half of this question, if you don't know the answer to its first half. This question should take you LESS THAN 10 minutes to answer, if you know the correct formula. (This is because it involves applying three times a formula whose arithmetic calculation takes only 2 minutes to do, for each of its three applications).

The last two parts of this question will concern Part A's Separate Chaining methodology. *These parts are not hard,* but they differ from Parts A-D by involving material that I did not cover in class. I suggest that you do not spend more than 15 minutes time on Parts E and F and *not worry* if they cause you difficulty. These parts are designed to give you extra credit if you have difficulties elsewhere in the test. You will not be penalized if you do not answer Parts E and F correctly (assuming the remainder of your test is fine).

E) All hash searches have "two" expected times associated with their search processes because the average time to doing a "successful" search (for a key stored in the file) is different from doing an "unsuccessful" search (for a key that is not stored in the file). Assume that the B-th address has $J_B \geq 1$ records stored in its separate chain. Then please tell me what is the expected time to do a both successful and an unsuccessful search into this bucket. (Hint: The two quantities are *trivially easy* to calculate, and they are distinctly different numbers. Also, your answer to this question should DEFINITELY be NO MORE than 2-5 sentences long. This is because I really just want to see two formulas that are either correct or incorrect.)

F) Let N denote the number of records stored in your hash file, and S denote its number of slots. In hash theory the "density" quantity α is the following ratio.

$$\alpha = \frac{N}{S} \tag{1}$$

In this notation, please do the following:

F-1 Tell me what is the average amount of time to do a unsuccessful search in a hash file as function of α. You can assume

1. All bucket-addresses are searched with equal probability
2. If a bucket address has zero records stored in it then it takes one unit of time less to search than a bucket containing one key (a quantity that you should have calculated when doing the half of Part-E that involves unsuccessful searches).

Your answer can be as simple as a one sentence long because I just want the correct formula. (I am not sure that I will give partial credit for the wrong answer, and I am simply will not attempt to give partial credit for a wrong answer that is more than 2-4 sentences long.)

F-2 This part is the analog of F-1 that asks you to instead calculate the expected time for a successful search. This part is also not hard, but it is slightly more challenging that Part F-1. Your analysis should use the following assumptions.

1. The probability of searching a bucket address B that contains J_B keys is $\frac{J_B}{N}$.
2. The amount of time used to search such a bucket was calculated by you in the part of Question E that considered "successful" searches.

Using Items (1) and (2), you should be able to express the expected time of your search as a summation formula. *Please write* down that very *simple* summation formula. Then after writing down that formula, you should take the limit of it as S approaches infinity (and as Equation (1) implies $N = \alpha S$) to calculate the value of this limit as solely a function of α.

Hint: This answer should not take you more than 2-5 minutes to derive when you see the trick, and you are wasting your time if you don't see the quite straightforward idea. Also your *entire answer* should **must not be more** than half-a-dozen sentences long, since I am looking only for correct answers and not intending to read long-winded answers for what is basically a simple idea.

AGAIN, I REMIND ALL STUDENTS not too waste too much time on Parts E and F of this question because they are extra credit questions, and a student will pass the 503 exam if he/she ignores these parts and does well on the remainder of the 503 half of PhD exam. (Also, they are not worth very many points)