CISC 621 Algorithms, Midterm exam
March 24, 2016

Name: _____

Multiple choice and short answer questions count 4 points each.

1. The algorithm we studied for median that achieves worst case linear time performance has which of the following properties?

   (a) Handles the more general problem of selecting the rank k item (median is the rank n/2 item for array of size n).

   (b) Makes a recursive call on an array of size n/5

   (c) Makes a recursive call on an array of size about 7n/10

   (d) all of the above

   [d]

2. The algorithm we studied for max-and-min that achieves worst case number of comparisons about 3n/2 has which of the following properties?

   (a) Starts with n/2 comparisons to form a set of "winners" and a set of "losers".

   (b) Provably achieves the minimal worst case number of comparisons possible.

   (c) both of the above

   (d) none of the above

   [c]

3. (5 points)

   Circle the sorting algorithms that run in worst case cost $\Theta(n \log(n))$.

   heapSort insertionSort introspectiveSort mergeSort quickSort

   [heapSort, introspectiveSort, mergeSort.]

4. (5 points)

   Circle the sorting algorithms that are not in place. (An *in place* algorithm uses a constant amount of memory other than the input data array).

   heapSort insertionSort introspectiveSort mergeSort quickSort

   [mergeSort]

5. (5 points)

   Circle the sorting algorithms that are randomized and have expected case cost $\Theta(n \log(n))$.

   heapSort insertionSort introspectiveSort mergeSort quickSort

   [introspectiveSort, quickSort]

6. Circle the sorting algorithms that can effectively use InsertionSort on base case arrays of size less than 16.

   heapSort introspectiveSort mergeSort quickSort

   [introspectiveSort, mergeSort, quickSort]

7. Both classical polynomial multiplication and Karatsuba's can be understood as divide and conquer algorithms working with polynomials divided in half as $f(x) = f_1(x)x^{n/2} + f_0(x)$, f has n terms and f_1, f_2 have $n/2$ terms. Regarding the number multiplications of halves (polynomials of $n/2$ terms, degree $n/2 - 1$) used, which is true?

   (a) It leads to the recurrence relation for cost of Karatsuba of $T(n) = 3T(n/2) + cn^{\lg(3)}$

(b) Classical uses 3 multiplications and Karatsuba uses 2.

(c) Classical uses 4 multiplications and Karatsuba uses 3.

(d) Classical uses 8 multiplications and Karatsuba uses 7.

[c]

8. Which is true of a binary min-heap with n elements?

   (a) The key at the root is greatest in the heap.

   (b) The heap is a (left) complete binary tree.

   (c) The heap is stored in an array A, and the last position, A[n], holds greatest entry.

   (d) none of the above

   [b]

9. Which priority queue implementation is efficient, worst case O($\log(n)$), for the operations of merging heaps (union) and splitting them in half?

   (a) binary heap

   (b) binomial heap

   (c) both binary and binomial heaps

   (d) neither binary nor binomial heaps

   [b]

10. Dynamic arrays and linked lists are alternate strategies on which to base many data structures. Fill in the blanks with "worst case", "amortized", or "expected" as appropriate.

    (a) A new element can be added at the front of a linked list in $\Theta(1)$ _____ time.
    [worst case]

    (b) A new element can be added at the end of a dynamic array (dynamic table) in $\Theta(1)$ _____ time.
    [amortized]

    (c) For any position i of a dynamic table, the i-th element can be accessed in $\Theta(1)$ _____ time.
    [worst case]

    (d) What is the cost of accessing the i-th element of a linked list (use Θ)? _____
    [$Theta(i)$]

11. A left leaning red-black tree implements the 2-3-4 tree balancing scheme as a binary search tree with reasonable balance. Which statement is true about LLRBT?

    (a) Every path from root to leaf has the same number of black edges.

    (b) If the edge from a node to it's left child is red then the edge from the node to it's right child is also red.

    (c) Both of the above

    (d) None of the above

    [a]

12. What condition must be considered when applying the Master theorem to a recurrence of the form $T(n) = aT(n/b) + cn^d$

    (a) A comparison of $n^{\log_a(b)}$ and n^d

    (b) A comparison of $n^{\log_a(b)}$ and $n^{\log_c(d)}$.

    (c) A comparison of $n^{\log_b(a)}$ and $n^{\log_c(d)}$.

    (d) A comparison of $n^{\log_b(a)}$ and n^d

    [d]

Answers will be graded for clarity and thoroughness as well as correctness.

13. (10 points) Select one of the following two algorithms and explain why it correctly solves the corresponding algorithmic problem P, Make clear the specification or invariant that lies at the heart of your argument.

(a) Problem: polynomial multiplication
Input: two polynomials f(x), g(x) of degree less than n, a power of 2.
Output: product h(x) of the two polynomials (of degree less than 2n).
Algorithm: Karatsuba(f,g,n)

1. If n = 1 return $h_0 = f_0 * g_0$.
2. Separate the upper $n/2$ coefficients and the lower $n/2$ coefficients of f:
   $f(x) = f^{(h)}(x)x^{n/2} + f^{(l)}(x)$.
   For example, if $n = 4$ and $f(x) = f_3x^3 + f_2x^2 + f_1x + f_0$,
   then $f^{(h)} = f_3x + f_2$ and $f^{(l)} = f_1x + f_0$.
   Similarly, $g(x) = g^{(h)}(x)x^{n/2} + g^{(l)}(x)$.
3. $P(x) = f^{(l)}(x)g^{(l)}(x)$
   $Q(x) = (f^{(h)}(x) + f^{(l)}(x))(g^{(h)}(x) + g^{(l)}(x))$
   $R(x) = f^{(h)}(x)g^{(h)}(x)$
4. return $R(x)x^n + (Q(x) - P(x) - R(x))x^{n/2} + P(x)$. Note that the multiplication by $x^n, x^{n/2}$ just describes shifting. For instance R_i is added to coefficient h_{i+n}.

[The result of polynomial multiplication, expressed in terms of high and low parts is $f^{(h)}(x)g^{(h)}(x)x^n + f^{(h)}(x)g^{(l)}(x) + f^{(l)}(x)g^{(h)}(x)x^{n/2} + f^{(l)}(x)g^{(l)}(x)$. Since $R(x)$ and $P(x$ are the first and third terms of this sum, it remains to show the middle term is correct. Here

$$\begin{aligned} Q - P - R &= (f^{(h)}(x) + f^{(l)}(x))(g^{(h)}(x) + g^{(l)}(x)) - f^{(l)}(x)g^{(l)}(x) - f^{(h)}(x)g^{(h)}(x) && (1)\\ &= f^{(h)}(x)g^{(h)}(x) + f^{(h)}(x)g^{(l)}(x) + f^{(l)}(x)g^{(h)}(x) + f^{(l)}(x)g^{(l)}(x) - f^{(h)}(x)g^{(h)}(x) - f^{(l)}(x)g^{(l)}(x) && (2)\\ &= f^{(h)}(x)g^{(l)}(x) + f^{(l)}(x)g^{(h)}(x) && (3) \end{aligned}$$

as required.
]

(b) Problem: x = Select(A, n, k)
Input: Unordered array of numbers A of size n and an index k.
Output: the entry x of A of rank k (x would be in position k if A were sorted)
Algorithm: x = RandomizedSelect(A, n, k)

1. If $(n = 1)$ return $A[1]$ // remark: k must be 1.
2. p = Partition(A, n);
3. If $(p = k)$ return $A[p]$ 4. If $(p > k)$ return RandomizedSelect$(A, p - 1, k)$; else return RandomizedSelect$(A + p, n - p, k - p)$;

You may assume Partition works correctly, but state the specification of what Partition does.

[Partition returns an index p and reorders the elements of A so that items in position 1..p-1 are less than A[p] and items in positions p+1..n are greater than A[p]. In particular, then, A[p] has rank p, and is thus the correct value to return when p = k (line 3).

Correctness proof is by induction on n. If n = 1 then k (which must be a valid index) is 1 and the one entry of A has rank 1 and is the correct value to return.

Inductive step: assume correct for array sizes less than n.
Case 1, $k = p$: (already discussed)
Case 2, $k < p$: The rank k item of (A,n) is the rank k item of (A, p-1), since these are precisely the p-1 items of rank less than p. Thus the recursive call in line 4 is correct.
Case 3, $k > p$: The rank k item of (A,n) is the rank k-p item of (A+p, p-1), since the p ignored items are those of the first 1..p ranks in (A,n). It suffices to find

the rank k-p item among the remaining n-p larger items. Thus the recursive call in line 5 is correct.]

14. (9 points)The Union-Find data structure for the Dynamic Disjoint Sets abstraction works on length n arrays boss and rank, such that initially boss[i]=i and rank[i] = 0.

    (a) Give pseudo code for the operations Union(a,b) and Find(a) such that Union assumes a and b are CEO's. It forms the merger of the two trees using the union by rank heuristic. Find(x) returns the CEO(i.e., root) of x's tree and performs path compression.

    [see unionfind handout. There was an error in find() there (it had the return statement of both with and without path compression, without first – and meant to be commented out. As a result it didn't do path compression. Sorry, no credit for verbatim copy of that mistake.]

    (b) Prove that with union by rank that for each node, height(a) $\leq$ rank[a] so that worst case cost of find is O(log(n))..

    [proof is by induction on tree size. Initialization has one node of height=rank=0, so the condition is valid in the base case. Union(a,b) changes the tree size by either making either a child of b or vice versa. If a is made child of b because rank(a) ¡ rank(b), then depth of all nodes in a is increased by 1. However, height(a) ¡= rank(a) by inductive hypothesis and rank(a) ¡ rank(b) assumption. Thus depth of nodes of a in new tree rooted at b is no more than existing height of b and the height ¡ rank property is preserved. The argument if rank(b) ¡ rank(a) is similar. Now consider the case rank(a) = rank(b). The height may actually be increased by 1, but the rank is explicitly increased by one, so the condition height ¡= rank is certainly preserved.]

    (c) State (without explanation) a better amortized cost for each find operation over the course of a program when path compression is used.

    [log(log(n)) or $\log * (n)$ or inverse Ackerman function of n]

15. (30 points) Select **two** of the following four algorithms. Describe the algorithm. State and explain the worst case runtime of the algorithm on inputs of size n. If randomization is involved, mention the expected runtime. If answer is given for more than two parts, only the first two will be read.

    (a) Binomial-heap-extract-min(H)

    [It costs O(lg(n)) worst case time. The method is to find the binomial tree with the min at root and chop off that root. The children of that tree constitute a heap (in reverse order). Merge the rest of the heap with that children heap. Merging the two heaps into order sorted by tree size costs O(lg(n)). then a consolidation pass does up to lg(n) "carry" operations costing another O(Lg(n)).]

    (b) BST-next(x), algorithm to find the node with the next larger key after x's key in a binary search tree.

    [Let h be the height of the search tree. Next(x) costs O(h), since you either go down to the leftmost node in the right subtree or (if right subtree is null) go up toward the root until the step in which the parent is to the right.]

    (c) Binary-heap-insert(H,k), algorithm to insert key k in a binary min heap H, and restore the heap property.

    [It costs O(lg(n)), since the method it to put the new element in last postion (of left complete binary tree stored in an array) and then "bubble up" to restore the heap property. Bubbling up can involve a swap at each node on the path from the new item to the root, thus up to O(lg(n)) steps.

```
bubble-up(H,i) {
    parent = floor(i/2);
    if i > 1  and H(i) < H(parent) then {
        swap(H(i), H(parent));
        bubble-up(H, parent);
    }
}
```

(d)
```
IntroSort(A,n,d){\\
1. if (n < 20) { InsertionSort(A,n); return; }
2. if (d <= 0) { HeapSort(A,n); return; }
3. p = Partition(A,n);
4. IntroSort(A, p, d-1); IntroSort(A+p, n-p, d-1); return;
}
```

Include explaination of the role of d and what it's initial value should be. You may assume that InsertionSort runs in worst case time $O(n^2)$, HeapSort in worst case time $O(n \log(n))$, Partition in worst case time $O(n)$.

[IntrospectiveSort(A,n) works by calling IntroSort(A,n,D) where D is about 2lg(n), in any case O(lg(n)). The recursion tree is like that of QuickSort, except that at depth D it switches to HeapSort. As in QuickSort, at each level of the recursion tree partitioning is done on portions of the array whose lengths add up to at most n, thus the quicksort style part of the computation has cost bounded by O(nD = O(nlg(n)). Suppose k portions of the array are to be sorted at depth D of the recursion tree, with sizes $n_1, n_2, \ldots, n_k$. HeapSort is applied to them at total cost $\sum_i n_i \lg(n_i)$. Since $\sum_i n_i < n$ and $\lg(n_i) < \lg(n)$, we have $\sum_i n_i \lg(n_i) < \lg(n) \sum_i n_i \leq \lg(n)n$.