# CS 373: Combinatorial Algorithms, Fall 2002

## Homework 2 (due Thursday, September 26, 2002 at 11:59:59 p.m.)

| Name: | | |
|---|---|---|
| Net ID: | Alias: | U G |

| Name: | | |
|---|---|---|
| Net ID: | Alias: | U G |

| Name: | | |
|---|---|---|
| Net ID: | Alias: | U G |

---

Homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since graduate students are required to solve problems that are worth extra credit for other students, **Grad students may not be on the same team as undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate or 1-unit grad student by circling U or G, respectively. Staple this sheet to the top of your homework. **NOTE: You must use different sheet(s) of paper for each problem assigned.**

---

## Required Problems

1. For each of the following problems, the input is a set of $n$ nuts and $n$ bolts. For each bolt, there is exactly one nut of the same size. Direct comparisons between nuts or between bolts are not allowed, but you can compare a nut and a bolt in constant time.

   (a) Describe and analyze a deterministic algorithm to find the largest bolt. *Exactly* how many comparisons does your algorithm perform in the worst case? *[Hint: This is very easy.]*

   (b) Describe and analyze a randomized algorithm to find the largest bolt. What is the *exact* expected number of comparisons performed by your algorithm?

   (c) Describe and analyze an algorithm to find the largest and smallest bolts. Your algorithm can be either deterministic or randomized. What is the *exact* worst-case expected number of comparisons performed by your algorithm? *[Hint: Running part (a) twice is definitely not the most efficient algorithm.]*

   In each case, to receive **full** credit, you need to describe the most efficient algorithm possible.

2. Consider the following algorithm:

```
SlowShuffle(A[1..n]) :
    for i ← 1 to n
        B[i] ←Null
    for i ← 1 to n
        index ← Random(1,n)
        while B[index] ≠ Null
            index ← Random(1,n)
        B[index] ← A[i]
    for i ← 1 to n
        A[i] ← B[i]
```

Suppose that Random($i$,$j$) will return a random number between $i$ and $j$ inclusive in constant time. SlowShuffle will shuffle the input array into a random order such that every permutation is equally likely.

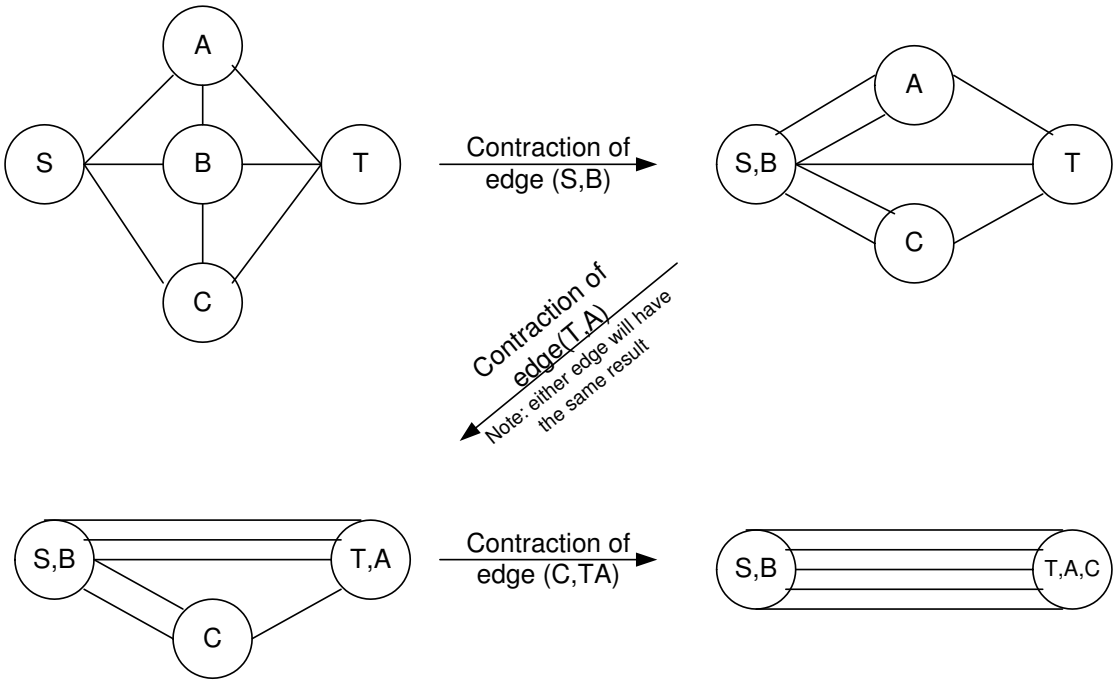
(a) What is the expected running time of the above algorithm. Justify your answer and give a tight asymptotic bound.

(b) Describe an algorithm that randomly shuffles an n-element array, so that every permutation is *equally* likely, in $O(n)$ time.

3. Suppose we are given an undirected graph $G = (V, E)$ together with two distinguished vertices $s$ and $t$. An **s-t min-cut** is a set of edges that once removed from the graph, will disconnect $s$ from $t$. We want to find such a set with the minimum cardinality(The smallest number of edges). In other words, we want to find the smallest set of edges that will seperate $s$ and $t$

To do this we repeat the following step $|V| - 2$ times: Uniformly at random, pick an edge from the set $E$ which contains all edges in the graph excluding those that directly connects vertices $s$ and $t$. Merge the two vertices that is connected by this randomly selected edge. If as a result there are several edges between some pair of vertices, retain them all. Edges that are between the two merged vertices are removed so that there are never any self-loops. We refer to this process of merging the two end-points of an edge into a single vertex as the *contraction* of that edge. Notice with each contraction the number of vertices of $G$ decreases by one.

As this algorithm proceeds, the vertex $s$ may get merged with a new vertex as the result of an edge being contracted. We call this vertex the $s$-vertex. Similarly, we have a $t$-vertex. During the contraction algorithm, we ensure that we never contract an edge between the $s$-vertex and the $t$-vertex.




(a) Give an example of a graph in which the probability that this algorithm finds an $s$-$t$ min-cut is exponentially small($O(1/a^n)$). Justify your answers.
*( Hint: Think multigraphs)*

(b) Give an example of a graph such that there are $O(2^n)$ number of $s$-$t$ min-cuts. Justify your answers.

4. Describe a modification of treaps that supports the following operations, each in $O(\log n)$ expected time:

   - INSERT($x$): Insert a new element $x$ into the data structure.
   - DELETE($x$): Delete an element $x$ from the data structure.
   - COMPUTERANK($x$): Return the number of elements in the data structure less than or equal to $x$.
   - FINDBYRANK($r$): Return the $k$th smallest element in the data structure.

   Describe and analyze the algorithms that implement each of these operations. *[Hint: Don't reinvent the wheel!]*

5. A *meldable priority queue* stores a set of keys from some totally ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue storing the empty set.
   - FINDMIN($Q$): Return the smallest element stored in $Q$ (if any).
   - DELETEMIN($Q$): Delete the smallest element stored in $Q$ (if any).
   - INSERT($Q, x$): Insert element $x$ into $Q$.
   - MELD($Q_1, Q_2$): Return a new priority queue containing all the elements stored in $Q_1$ and $Q_2$. The component priority queues are destroyed.
   - DECREASEKEY($Q, x, y$): Replace an element $x$ of $Q$ with a smaller key $y$. (If $y > x$, the operation fails.) The input is a pointer directly to the node in $Q$ storing $x$.
   - DELETE($Q, x$): Delete an element $x \in Q$. The input is a pointer directly to the node in $Q$ storing $x$.

   A simple way to implement this data structure is to use a heap-ordered binary tree, where each node stores an element, a pointer to its left child, a pointer to its right child, and a pointer to its parent. MELD($Q_1, Q_2$) can be implemented with the following randomized algorithm.

   - If either one of the queues is empty, return the other one.
   - If the root of $Q_1$ is smaller than the root of $Q_2$, then recursively MELD $Q_2$ with either right($Q_1$) or left($Q_1$), each with probability $1/2$.
   - Similarly, if the root of $Q_2$ is smaller than the root of $Q_1$, then recursively MELD $Q_1$ with a randomly chosen child of $Q_2$.

   (a) Prove that for *any* heap-ordered trees $Q_1$ and $Q_2$, the expected running time of MELD($Q_1, Q_2$) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: How long is a random path in an $n$-node binary tree, if each left/right choice is made with equal probability?]* For extra credit, prove that the running time is $O(\log n)$ *with high probability.*

   (b) Show that each of the operations DELETEMIN, INSERT, DECREASEKEY, and DELETE can be implemented with one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ with high probability.)

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   The following randomized algorithm selects the $r$th smallest element in an unsorted array $A[1..n]$. For example, to find the smallest element, you would call RANDOMSELECT($A, 1$); to find the median element, you would call RANDOMSELECT($A, \lfloor n/2 \rfloor$). Recall from lecture that PARTITION splits the array into three parts by comparing the pivot element $A[p]$ to every other element of the array, using $n - 1$ comparisons altogether, and returns the new index of the pivot element.

```
RandomSelect(A[1..n], r):
    p ← Random(1, n)
    k ← Partition(A[1..n], p)

    if r < k
        return RandomSelect(A[1..k−1], r)
    else if r > k
        return RandomSelect(A[k+1..n], r−k)
    else
        return A[k]
```

(a) State a recurrence for the expected running time of RandomSelect, as a function of both $n$ and $r$.

(b) What is the *exact* probability that RandomSelect compares the $i$th smallest and $j$th smallest elements in the input array? The correct answer is a simple function of $i$, $j$, and $r$. *[Hint: Check your answer by trying a few small examples.]*

(c) Show that for any $n$ and $r$, the expected running time of RandomSelect is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, find the *exact* expected number of comparisons, as a function of $n$ and $r$.

(d) What is the expected number of times that RandomSelect calls itself recursively?

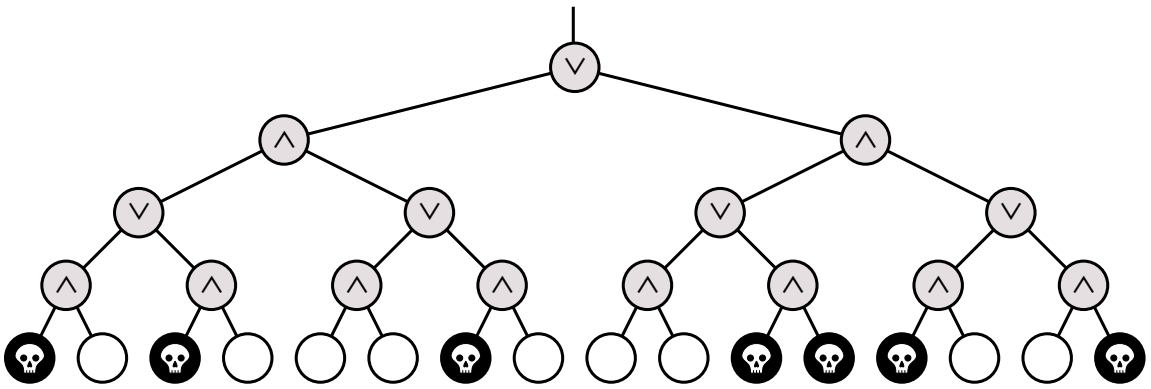
## Practice Problems

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with $4^n$ leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.

   You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are *and* gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

   (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

   (b) Unfortunately, Death won't let you even look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $\Theta(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*



2. WHat is the *exact* number of nodes in a skip list storing n keys, *not* counting the sentinel nodes at the beginning and end of each level? Justify your answer.

3. Suppose we are given two sorted arrays $A[1..n]$ and $B[1..n]$ and an integer $k$. Describe an algorithm to find the $k$th smallest element in the union of $A$ and $B$. (For example, if $k = 1$, your algorithm should return the smallest element of $A \cup B$; if $k = n$, our algorithm should return the median of $A \cup B$.) You can assume that the arrays contain no duplicates. Your algorithm should be able to run in $\Theta(\log n)$ time. *[Hint: First try to solve the special case $k = n$.]*