• Homework 2
  • Due Saturday at 2am (so, basically Friday night)

• Midterm: Tuesday, March 10, 2015
  • 15% of your grade
  • In class
  • Closed book
  • Written
  • There will be no homework on March 13
Let L denote an array of integers, and let f have be a function with prototype void f(int[] arr).

**Exercise:** How can you further reduce the search space for the 8 Queens problem?

**Exercise:** Give recursive code that will process every subset of L and count how many have sum k.
/Call initially with parameters (L, 0, 0)
static int countSubsets(int[] L, int idx, int sum, int k)
{
    if (idx == L.length) return sum == k ? 1 : 0;
    return countSubsets(L, idx+1, sum, k)
        + countSubsets(L, idx+1, sum+L[idx], k);
}
Let L denote an array of integers, and let f have be a function with prototype void f(int[] arr).

**Exercise:** Give recursive code that will call the function f one time for each permutation of L (can assume entries are distinct).
Let L denote an array of integers, and let f have be a function with prototype void f(int[] arr).

**Exercise:** Give recursive code that will call the function f one time for each permutation of L (can assume entries are distinct).

**Exercise:** Give iterative code for calling f once for each permutation of L in lexicographic order.
Let $L$ denote an array of integers, and let $f$ have be a function with prototype `void f(int[] arr)`.

**Exercise:** Give recursive code that will call $f$ once for each subsequence of $L$ of length 4.

**Exercise:** Give iterative code that will call $f$ once for each subsequence of $L$ of length 4.
To determine whether complete search will solve your problem, address the following:

- How can I traverse the search space?
- **How long with the complete search take?**
- How can I implement the complete search?
• To determine whether a greedy algorithm will solve your problem, address the following:
  • **What is my greedy choice function?**
  • How long will the greedy algorithm take?
  • **Is it correct?**
  • How can I implement my greedy algorithm?
You are given a list of tasks, with task $i$ requiring time interval $[a_i, b_i)$, and the requirement that no two tasks can be scheduled at the same time. Give an algorithm to compute the largest number of tasks that can be completed.
Sort tasks in ascending order by $b_i$ breaking ties by ascending $a_i$-values. Always choose the earlier task in the sorted order that is feasible. Now we must show this is correct. Let $P(t)$ denote the maximum number of tasks that can be assigned using times in $[t, \infty)$. Note that $P(s) \geq P(t)$ if $s \leq t$. Suppose we are choosing between $[a_i, b_i)$ and $[a_j, b_j)$ as our earlier task with $b_i \leq b_j$. Then $P(b_i) \geq P(b_j)$ making the first interval at least as good.
You are given a list of objects, with object $i$ having weight $w_i$ and value $v_i$. Assuming you can take fractional amounts of each object, and your sack has capacity $C$, compute the largest value you can carry.
Sort tasks in descending order by $v_i/w_i$, and then use objects in order taking as much as possible of each. To show this is optimal, suppose $v_i/w_i \geq v_j/w_j$, you have taken a positive amount of object $j$, and you have not taken all of object $i$. Then trading object $j$ for object $i$ cannot disimprove the total value.
You are given a list of tasks, with task $i$ having duration $d_i$ in days. For each day before task $i$ is started, a cost $s_i$ is paid. Choose an ordering of the tasks that minimizes the total cost paid.
Sort tasks in descending order by $s_j/d_j$. To see this is optimal, suppose you have an ordering of the tasks with task $i$ immediately preceding task $j$. The change in value of swapping these tasks is $s_id_j - s_jd_i$. This is non-negative precisely when $s_i/d_i \leq s_j/d_j$. 
Give a greedy algorithm for computing the minimum/maximum spanning tree
We do the minimum case. Sort the edges by weight. Only add an edge if it doesn’t create a cycle in the forest of edges already added. Suppose this algorithm is not optimal on some input. Then there must be some first edge $e$ added such that the resulting forest cannot be completed into a minimum spanning tree, but there is a MST $T$ that can be constructed from the edges added before $e$. Then adding $e$ to $T$ creates a cycle, where at least one edge in the cycle has value greater than or equal to $e$ (as adding $e$ didn’t create a cycle in our algorithm). Adding $e$ and removing this larger valued edge creates a tree that is at least as good. Implementation details will be discussed when we cover graph algorithms.
Given infinitely many coins with denominations in the set \{50, 25, 10, 5, 1\}, give an algorithm for making change for the amount \(A\) that uses the least number of coins.
Sort the coins in descending order, and use as many as possible of each coin before moving to the next coin. Now we prove this is optimal. Let $50 = c_1 > \cdots > c_5 = 1$ be the coin values. Assume we have created change for the amount $A$ only using coins strictly smaller than $c_i$, where $A \geq c_i$. If some of the coins used total to $c_i$, then we can replace them with $c_i$ and obtain a better result. Otherwise we have the following cases:

(a) $c_i = 50$ and a quarter, and 3 dimes are used: replace with a 50 cent piece and a nickel.

(b) $c_i = 25$ and 3 dimes are used: replace with a quarter and a nickel.
• Divide and conquer:
  • Break a problem into multiple smaller subproblems
  • Solve these subproblems
  • Combine the results

• Very popular:
  • Sorting (quicksort, mergesort)
  • Searching (binary search)
• Given a list of \( n \leq 30 \) signed integers and a 64-bit integer \( L \), determine if some sublist (not necessarily contiguous) sums up to the \( L \).
  • Split list in half, compute all possible sums that can occur in each half
  • \( O(2^{n/2} \log(2^{n/2})) = O(n^{2n/2}) \)
• Multiply two big integers: \( a, b \) of length \( 2m \) bits
  • Note: \( a = a_h \cdot 2m + a_l, b = b_h \cdot 2m + b_l \)
  • Therefore \( ab = a_h b_h 2^{2m} + (a_h b_l + a_l b_h)2^m + a_l b_l \)
  • Note that \( a_h b_l + a_l b_h = (a_h + a_l)(b_h + b_l) - a_h b_h - a_l b_l \)
  • Thus, \( ab \) can be computed recursively using 3 multiplications with operands half as big
  • Running time: \( O(n^{\log 3}) \), where \( \log 3 \approx 1.58 \)
  • Karatsuba’s algorithm
• Standard use: search for an element in a sorted array
  • Arrays.binarySearch, Collections.binarySearch in Java
  • lower_bound, upper_bound, binary_search in C++
• Recall that Java’s binarySearch doesn’t guarantee that the first occurrence of an element will be returned (when searching a list with duplicate elements)
• Still a popular interview problem!
Useful application of binary search: remove one dimension of a complex problem

**Example**: you are given a list of events that occur, and the time at which it occurs during your trip through the dessert

- Spring a leak, causing you to lose gas at 1 unit per minute (multiple gas leaks accumulate)
- Find a mechanic (fixes all leaks)
- Find a gas station (fills tank)
- Reach end of trip

What is the minimum gas tank size required for the journey?
Let L be an array of integers.

**Exercise:** Compute the maximum value in L using divide and conquer.

**Exercise:** Assuming L is sorted in increasing order, implement binary search to find the first occurrence of k in L, and a second implementation to find the last occurrence.
Let \( L \) be an array of integers.

**Exercise**: Given \( k \), compute the \( k \)th (0-based) smallest element of \( L \).

**Exercise**: Assuming \( L \) has nonnegative entries, partition \( L \) into \( k \) contiguous subintervals so that the maximum sum over any of the subintervals is minimized.