9/11/07 Name: _____

1) Consider the following backtracking algorithm for solving the partial digest problem (PDP).

```
PartialDigest(L)
1       width ← Maximum element in L
2       DELETE(width, L)                    // DELETE(e,Z) removes e from Z
3       X ← {0, width}
4       Place(L, X)

Place(L, X)
1       if L is empty
2           output X
3           return
        end if
4       y ← Maximum element in L
5       if Δ (y, X) ⊆ L                     // Δ (y, X) calculates multiset of distances from
6           Add y to X and remove lengths Δ (y, X) from L   //   point y to all points in set X
7           Place(L, X)
8           Remove y from X and add lengths Δ (y, X) to L
        end if
9       if Δ (width-y, X) ⊆ L
10          Add width-y to X and remove lengths Δ (width-y, X) from L
11          Place(L, X)
12          Remove width-y from X and add lengths Δ (width-y, X) to L
        end if
```

Let |X| = n. (the X for the final answer)

a) In the worst case, how many nodes would the "search space" tree for this algorithm contain?

b) How would you compute Δ (y, X)?

c) If n is the size of X, what is an upper bound (big-oh) on the work to calculate Δ (y, X)?

d) What would be the worst case big-oh for this algorithm?

e) In the best case, we can prune one of the children of each node because it is not feasible. How many nodes would the "search space" tree for this algorithm contain?

f) What would be the best case big-oh for this algorithm?

g) For real data, would you expect the average case big-oh to be?

2) In general the recursive backtracking algorithm (for optimization problems) look something like:

```
Backtrack( treeNode n ) {
    treeNode c;
    for each child c of n do
            if promising(c) then
                    if c is a solution that's better than best then
                            best = c
                    else
                            Backtrack(c)
                    end if
            end if
        end for
} // end Backtrack
```

General Notes about Backtracking:

- the depth-first nature of backtracking only stores information about the current branch being explored so the memory usage is “low”
- Each node of the search space tree maintains the state of a partial solution. In general the state consists of potentially large arrays that change little between parent and child. To avoid having multiple copies of these arrays, a single “global” state is maintained which is updated before we go down to the child (via a recursive call) and undone when we backtrack to the parent.

a) In the PartialDigest algorithm, why do we not have a “for loop” to iterate through the children?

b) For the PartialDigest algorithm, where (what line numbers) do we go from the parent to the child node?

c) For the PartialDigest algorithm, where (what line numbers) do we update the single “global” state before going down to the child node?

d) For the PartialDigest algorithm, where (what line numbers) do we undo the single “global” state when we backtrack to the parent node?