Suggested Solutions to Midterm Problems

Problems

1. Construct an open gray code of length $\lceil \log_2 19 \rceil$ $(= 5)$ for 19 objects. Please describe how the gray code is constructed *systematically* from gray codes of smaller lengths.

   Solution.
   Let $(c_1, c_2, \ldots, c_n)^R$ denote the list $c_n, c_{n-1}, \ldots, c_1$.

   $19 = 2 \times 9 + 1$; $9 = 2 \times 4 + 1$; $4 = 2 \times 2$. So, we will start with building a code for 2 objects and then codes for 4, 8, 9, 18, and finally 19 objects.

   Code of length 1 for 2 objects: $0, 1$.
   Code #1 of length 2 for 2 objects: $00, 01$.
   Code #2 of length 2 for 2 objects: $10, 11$.
   Code of length 2 for 4 objects: $00, 01, (10, 11)^R$.
   Code of length 2 for 4 objects: $00, 01, 11, 10$.
   Code #1 of length 3 for 4 objects: $000, 001, 011, 010$.
   Code #2 of length 3 for 4 objects: $100, 101, 111, 110$.
   Code of length 3 for 8 objects: $000, 001, 011, 010, (100, 101, 111, 110)^R$.
   Code of length 3 for 8 objects: $000, 001, 011, 010, 110, 111, 101, 100$.
   Code of length 4 for 9 objects: $0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, \mathbf{1100}$. (open)
   Code #1 of length 5 for 9 objects: $00000, 00001, 00011, 00010, 00110, 00111, 00101, 00100, \mathbf{01100}$.
   Code #2 of length 5 for 9 objects: $10000, 10001, 10011, 10010, 10110, 10111, 10101, 10100, \mathbf{11100}$.
   Code of length 5 for 18 objects: $00000, 00001, 00011, 00010, 00110, 00111, 00101, 00100, 01100,$ $(10000, 10001, 10011, 10010, 10110, 10111, 10101, 10100, 11100)^R$.
   Code of length 5 for 18 objects: $00000, 00001, 00011, 00010, 00110, 00111, 00101, 00100, 01100,$ $11100, 10100, 10101, 10111, 10110, 10010, 10011, 10001, 10000$.
   Code of langth 5 for 19 objects: $00000, 00001, 00011, 00010, 00110, 00111, 00101, 00100, 01100,$ $11100, 10100, 10101, 10111, 10110, 10010, 10011, 10001, 10000, \mathbf{11000}$. (open)
   □

2. Consider the following program segment in the celebrity algorithm.

```
i := 1;
j := 2;
next := 3;
while next <= n+1 do
   if Know[i,j] then i:= next
   else j := next;
```

```
      next := next + 1;
      end;
   if i = n+1 then candidate := j
   else candidate := i;
```

(a) Find an appropriate loop invariant for the while loop that is sufficient to show that `candidate` will be the only possible candidate for the celebrity after the execution of the segment.

Solution. An appropriate loop invariant is "if k is the celebrity, then $k = i$, $k = j$, or $next \leq k \leq n$" (plus $1 \leq i \leq next$, $2 \leq j \leq next$, $3 \leq next \leq n + 2$, which is omitted for brevity). □

(b) Prove that the loop invariant found above is indeed a loop invariant.

Solution. We need to show that (1) the assertion is true at the beginning of the loop and (2) given that the assertion is true and the condition of the while loop holds, the assertion will still be true after the loop body is executed.
(1) At the beginning of the loop, $i = 1$, $j = 2$, and $next = 3$. Apparently, if k is the celebrity, then $1 \leq k \leq n$ and hence $k = 1 = i$, $k = 2 = j$, or $next = 3 \leq k \leq n$.
(2) Now we assume that the assertion "if k is the celebrity, then $k = i$, $k = j$, or $next \leq k \leq n$" is true before the next iteration and the loop condition holds, i.e., $next \leq n + 1$. Let i', j', and $next'$ denote respectively the values of i, j, and $next$ after the iteration. We need to show that "if k is the celebrity, then $k = i'$, $k = j'$, or $next' \leq k \leq n$". From the loop body, we deduce the following relationship:

$$i' = \begin{cases} next & \text{if } Know[i, j] \\ i & \text{otherwise} \end{cases}$$

$$j' = \begin{cases} next & \text{if } \neg Know[i, j] \\ j & \text{otherwise} \end{cases}$$

$$next' = next + 1$$

There are two cases to consider: $Know[i, j]$ and $\neg Know[i, j]$. In the first case, i cannot be the celebrity. So, the truth of "if k is the celebrity, then $k = i$, $k = j$, or $next \leq k \leq n$" implies that of "if k is the celebrity, then $k = j$, or $next \leq k \leq n$", which is equivalent to "if k is the celebrity, then $k = j$, $k = next$, or $next + 1 \leq k \leq n$". Since $i' = next$, $j' = j$ and $next' = next + 1$, it follows that "if k is the celebrity, then $k = i'$, $k = j'$, or $next' \leq k \leq n$", which concludes the first case. The second case be carried out in an analogous manner. □

3. Find the asymptotic behavior of the function $T(n)$ defined as follows:

$$\begin{cases} T(1) = 1 \\ T(n) = 4T(n/2) + n^2, \quad n = 2^i \ (i \geq 1) \end{cases}$$

You should try to solve this problem without resorting to the general theorem for divide-and-conquer relations discussed in class. The asymptotic bound should be as tight as possible. (Hint: guess and verify by induction.)

Solution. We guess that $T(n) = O(n^2 \log n)$ (from "vague memory" of Theorem 3.4 in Manber's book or after having tried $O(n^2)$ and $O(n^3)$). We verify this by an inductive proof that $T(n) \leq 2n^2 \log n$, for all $n = 2^i \geq 2$ (i.e., we are taking N to be 2 and c also to be 2 in the definition of O).

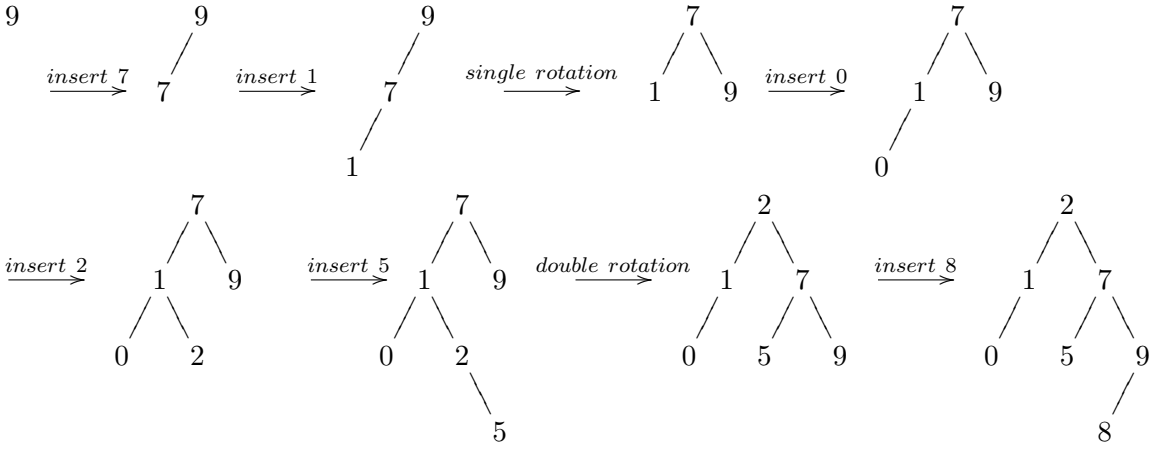
Base case ($n = 2$): $T(2) = 4T(1) + 2^2 = 8 \leq 2 \times 2^2 \log 2$.
Inductive step ($2n = 2^i, i > 1$):

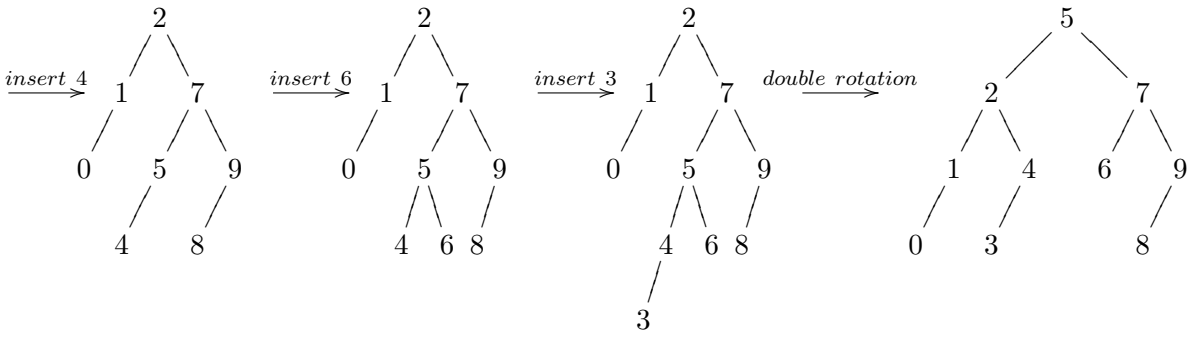
$$\begin{aligned} T(2n) &= 4T(n) + (2n)^2 \\ &\leq 4(2n^2 \log n) + 4n^2 \quad \text{(from the induction hypothesis)} \\ &\leq 8n^2 \log n + 8n^2 \\ &= 2(2n)^2(\log n + 1) \\ &= 2(2n)^2(\log n + \log 2) \\ &= 2(2n)^2 \log 2n. \end{aligned}$$

□

4. Show all intermediate and the final AVL trees formed by inserting the numbers 9, 7, 1, 0, 2, 5, 8, 4, 6, and 3 (in this order) into an empty tree. Please use the following ordering convention: the key of an internal node is larger than that of its left child and smaller than that of its right child. If re-balancing operations are performed, please also show the tree before re-balancing and indicate what type of rotation is used in the re-balancing.

Solution.





□

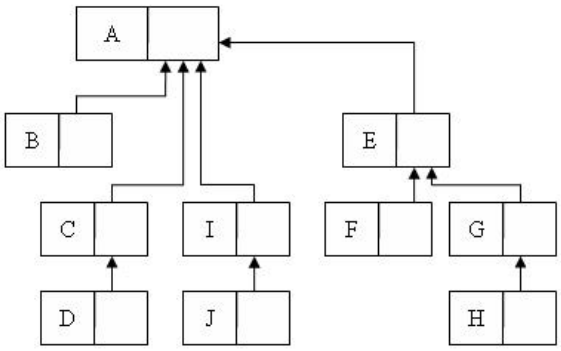
5. Consider solutions to the union-find problem discussed in class. Suppose we start with a collection of ten elements: A, B, C, D, E, F, G, H, I, and J.

   (a) Assuming the balancing, but not path compression, technique is used, draw a diagram showing the grouping of these ten elements after the following operations are completed:

   i. union(A,B)
   ii. union(C,D)
   iii. union(E,F)
   iv. union(G,H)
   v. union(I,J)
   vi. union(A,D)
   vii. union(F,G)
   viii. union(D,J)
   ix. union(D,H)

   In the case of combining two groups of the same size, please always point the second group to the first.

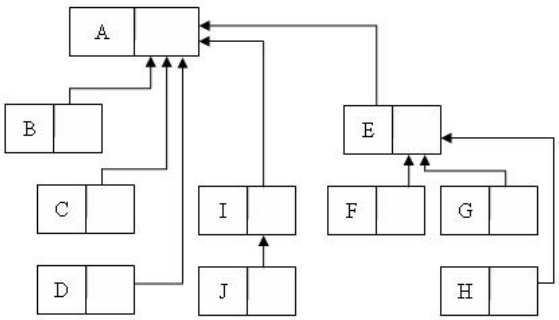
   Solution.



□

(b) Repeat the above, but with both balancing and path compression.

Solution.



□

6. The Knapsack Problem is defined as follows: Given a set S of n items, where the ith item has an integer size $S[i]$, and an integer K, find a subset of the items whose sizes sum to exactly K or determine that no such subset exists.

Below is an algorithm for determining whether a solution to the problem exists.

Algorithm Knapsack (S, K)**;**
begin
$P[0,0].exist := true;$
for $k := 1$ **to** K **do**
$P[0,k].exist := false;$
for $i := 1$ **to** n **do**
for $k := 0$ **to** K **do**
$P[i,k].exist := false;$
if $P[i-1,k].exist$ **then**
$P[i,k].exist := true;$
$P[i,k].belong := false$
else if $k - S[i] \geq 0$ **then**
if $P[i-1,k-S[i]].exist$ **then**
$P[i,k].exist := true;$
$P[i,k].belong := true$
end

(a) Design an algorithm to recover the solution recorded in the array P. (5 points)

Solution.

Procedure Print_Solution (S, P, n, K)**;**
begin

```
    if ¬P[n, K].exist then
        print "no solution"
    else i := n;
        k := K;
        while k > 0 do
            if P[i, k].belong = true then
                print i;
                k := k − S[i];
            i := i − 1
end
```

□

(b) Modify the given algorithm to solve a variation of the knapsack problem where each item has an unlimited supply. (10 points)

Solution. Insert "$P[0,0].belong := 0$;" after "$P[0,0].exist := true$;" and modify the last five lines before "end" as follows:

```
            P[i, k].belong := 0
        else if k − S[i] ≥ 0 then
                if P[i, k − S[i]].exist then
                    P[i, k].exist := true;
                    P[i, k].belong := P[i, k − S[i]].belong + 1
```

□

7. Consider rearranging the following array into a max heap using the bottom-up approach.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	7	3	5	9	14	8	11	6	4	10	15	13	12	2

Please show the result (i.e., the contents of the array) after a new element is added to the current collection of heaps (at the bottom) until the entire array has become a heap.

Solution.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	7	3	5	9	14	8	11	6	4	10	15	13	12	2
1	7	3	5	9	14	<u>12</u>	11	6	4	10	15	13	<u>8</u>	2
1	7	3	5	9	<u>15</u>	12	11	6	4	10	<u>14</u>	13	8	2
1	7	3	5	<u>10</u>	15	12	11	6	4	<u>9</u>	14	13	8	2
1	7	3	<u>11</u>	10	15	12	<u>5</u>	6	4	9	14	13	8	2
1	7	<u>15</u>	11	10	<u>14</u>	12	5	6	4	9	<u>3</u>	13	8	2
1	<u>11</u>	15	<u>7</u>	10	14	12	5	6	4	9	3	13	8	2
<u>15</u>	11	<u>14</u>	7	10	<u>13</u>	12	5	6	4	9	3	<u>1</u>	8	2

□

8. Prove that the sum of the heights of all nodes in a complete binary tree with n nodes is at most $n-1$. (A complete binary tree with n nodes is one that can be compactly represented by an array A of size n, where the root is stored in $A[1]$ and the left and the right children of $A[i]$, $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$, are stored respectively in $A[2i]$ and $A[2i+1]$. Notice that, in Manber's book a complete binary tree is referred to as a balanced binary tree and a full binary tree as a complete binary tree. Manber's definitions seem to be less frequently used. Do not let the different names confuse you.)

   Solution. Let $G(n)$ denote the sum of the heights of all nodes in a complete binary tree with n nodes. For a full binary tree (a special case of complete binary trees) with $n = 2^{h+1} - 1$ nodes where h is the height of the tree, we already know that $G(n) = 2^{h+1} - (h+2) = n - (h+1) \leq n-1$. With this as a basis, we prove the general case of arbitrary complete binary trees by induction on the number n (≥ 1) of nodes.

   Base case ($n = 1$ or $n = 2$): When $n = 1$, the tree is the smallest full binary tree with one single node whose height is 0. So, $G(n) = 0 \leq 1 - 1 = n - 1$. When $n = 2$, the tree has one additional node as the left child of the root. The height of the root is 1, while that of its left child is 0. So, $G(n) = 1 \leq 2 - 1 = n - 1$.

   Inductive step ($n > 2$): If n happens to be equal to $2^{h+1} - 1$ for some $h \geq 1$, i.e., the tree is full, then we are done; note that this covers the case of $n = 3 = 2^{1+1} - 1$. Otherwise, suppose $2^{h+1} - 1 < n < 2^{h+2} - 1$ ($h \geq 1$), i.e., the tree is a "proper" complete binary tree with height $h+1 \geq 2$. We observe that at least one of the two subtrees of the root is full, while the other is complete (possibly full). There are three cases to consider:

   Case 1: The left subtree is full with n_1 nodes and the right one is complete but not full with n_2 nodes (such that $n_1 + n_2 + 1 = n$). In this case, both subtrees much be of height h and $n_1 = 2^{h+1} - 1$. From the special case of full binary trees and the induction hypothesis, $G(n_1) = 2^{h+1} - (h+2) = n_1 - (h+1)$ and $G(n_2) \leq n_2 - 1$. $G(n) = G(n_1) + G(n_2) + (h+1) \leq (n_1 - (h+1)) + (n_2 - 1) + (h+1) = (n_1 + n_2 + 1) - 2 \leq n - 1$.

   Case 2: The left subtree is full with n_1 nodes and the right one is also full with n_2 nodes. In this case, the left subtree much be of height h and $n_1 = 2^{h+1} - 1$, while the right subtree much be of height $h-1$ and $n_2 = 2^h - 1$. From the special case of full binary trees, $G(n_1) = 2^{h+1} - (h+2) = n_1 - (h+1)$ and $G(n_2) = 2^h - (h+1) = n_2 - h$. $G(n) = G(n_1) + G(n_2) + (h+1) \leq (n_1 - (h+1)) + (n_2 - h) + (h+1) = (n_1 + n_2 + 1) - (h+1) \leq n - 1$.

   Case 3: The left subtree is complete but not full with n_1 nodes and the right one is full with n_2 nodes. In this case, the left subtree much be of height h, while the right subtree much be of height $h-1$ and $n_2 = 2^h - 1$. From the induction hypothesis and the special case of full binary trees, $G(n_1) \leq n_1 - 1$ and $G(n_2) = 2^h - (h+1) = n_2 - h$. $G(n) = G(n_1) + G(n_2) + (h+1) \leq (n_1 - 1) + (n_2 - h) + (h+1) = (n_1 + n_2 + 1) - 1 = n - 1$. □

9. Let $x_1, x_2, \cdots, x_{2n-1}, x_{2n}$ be a sequence of $2n$ real numbers. Design an algorithm to partition the numbers into n pairs such that the maximum of the n sums of pair is

minimized. It may be intuitively easy to get a correct solution. You must explain how the algorithm can be designed using induction. (15 points)

Solution. We first fix some notations:

- We represent a partition of a list $x_1, x_2, \cdots, x_{2n-1}, x_{2n}$ into n pairs as a set of sets of two elements $\{\{y_1, y_2\}, \{y_3, y_4\}, \cdots, \{y_{2n-1}, y_{2n}\}\}$, where $y_1, y_2, \cdots, y_{2n-1}, y_{2n}$ is a permutation of $x_1, x_2, \cdots, x_{2n-1}, x_{2n}$.
- For a list A of $2n$ elements, let **MinMaxPair**(A) denote some partition that meets the problem requirement for $2n$ elements.
- $A \setminus B$, where A is a list and B a set, denotes the list of elements in A but not in B. We stipulate that elements in $A \setminus B$ appear in the same order as in A.

We are given a list $X = x_1, x_2, \cdots, x_{2n-1}, x_{2n}$. If $n = 1$, i.e., there are only two elements, the solution is obvious, namely $\{\{x_1, x_2\}\}$. Now consider the cases of $n > 1$. Let y_1 denote the smallest element and y_{2n} the largest element in X. We claim that **MinMaxPair**$(X \setminus \{y_1, y_{2n}\}) \cup \{\{y_1, y_{2n}\}\}$ meets the problem requirement for $2n$ elements, i.e., the pair $\{y_1, y_{2n}\}$ is part of an optimal partition. Suppose $\{y_i, y_j\}$ is the pair with the largest sum in **MinMaxPair**$(X \setminus \{y_1, y_{2n}\})$. Pairing y_{2n} with either y_i or y_j (instead of y_1) would produce a pair whose sum is at least as large as that of $\{y_1, y_{2n}\}$ and that of any pair in **MinMaxPair**$(X - \{y_1, y_{2n}\})$. To compute **MinMaxPair**$(X \setminus \{y_1, y_{2n}\})$, we need to solve the same problem with two elements less and here we invoke the induction hypothesis.

In the above, we select and remove the smallest and the largest elements from the current list in each step. This would incur a complexity of $O(n)$ for each step, making the complexity of the whole algorithm $O(n^2)$. We can improve this by sorting the list right in the beginning before pairing up the elements. So, the algorithm can be summarized as follows.

(a) Sort the input list X to get Y.

(b) Suppose the current list $Y = y_1, y_2, \cdots, y_{2i-1}, y_{2i}$ $(i \geq 1)$. Remove and output the pair $\{y_1, y_{2i}\}$ from Y.

(c) Repeat the previpus step until Y is empty.

□