# Suggested Solutions to Midterm Problems

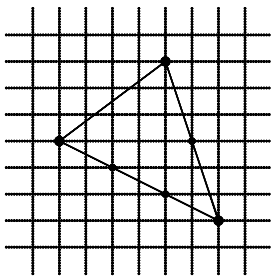
1. Construct a Gray code of length $\lceil \log_2 14 \rceil$ $(= 4)$ for 14 objects. Please explain how the Gray code is constructed *systematically* from Gray codes of smaller lengths.

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

   $14 = 2 \times 7$; $7 = 8 - 1$ (we are using reversed induction here); $8 = 2 \times 4$; $4 = 2 \times 2$. So, we will start with building a code for 2 objects and then codes for 4, 8, 7, and finally 14 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 3 for 7 objects: $000, 001, 011, 010, 110, 111, 101$. (open)
   Code #1 of length 4 for 7 objects: $0000, 0001, 0011, 0010, 0110, 0111, 0101$. (open)
   Code #2 of length 4 for 7 objects: $1000, 1001, 1011, 1010, 1110, 1111, 1101$. (open)
   Code of length 4 for 14 objects: $0000, 0001, 0011, 0010, 0110, 0111, 0101,$
   $(1000, 1001, 1011, 1010, 1110, 1111, 1101)^R$.
   Code of length 4 for 14 objects: $0000, 0001, 0011, 0010, 0110, 0111, 0101,$
   $1101, 1111, 1110, 1010, 1011, 1001, 1000$. $\square$

2. The *lattice* points in the plane are the points with integer coordinates. Let $T$ be a triangle such that all of its three vertices are lattice points; see the figure below. Let $p$ be the number of lattice points that are on the boundary of $T$ (including its vertices), and let $q$ be the number of lattice points that are inside $T$. Prove [*by induction*] that the area of $T$ is $\frac{p}{2} + q - 1$.

   *Solution.* The proof is by induction on $p + q$.

   Base case ($p + q = 3$): In this case, $p = 3$ and $q = 0$. A triangle satisfying this condition must have a side of unit length and the height with that side as the base must be one; otherwise, the triangle would either have more than three lattice points on the boundary or have at least one lattice point inside the triangle. Therefore, its area is $\frac{1}{2}(1 \times 1) = \frac{1}{2} = \frac{3}{2} + 0 - 1 = \frac{p}{2} + q - 1$.

Inductive step ($p + q > 3$): A triangle with $p + q > 3$ either (a) has at least one lattice point on the boundary that is not a vertex or (b) has at least one lattice point inside the triangle.

Case (a): Suppose the triangle has $p_1$, $p_2$, and $p_3$ lattice points on the three sides respectively and $q$ ($q \geq 0$) lattice points inside; so, $p_1 + p_2 + p_3 - 3 = p$. Without loss of generality, we assume that $p_3 > 2$ such that one of the $p_3$ lattice points is not a vertex. Connect the non-vertex lattice point to the opposite vertex with a line segment, to divide the triangle into two that share the line segment as a side. Suppose the new shared side has $p_4$ lattice points and the side with $p_3$ lattice points is divided into two smaller sides, each with $p_3'$ and $p_3''$ lattice points respectively. Clearly, $p_3' + p_3'' - 1 = p_3$, as the two smaller sides share a vertex. Now, one of the smaller triangles has $p_1$, $p_3'$, and $p_4$ lattice points on the three sides and the other has $p_2$, $p_3''$, and $p_4$ lattice points on the three sides. Suppose they respectively have $q'$ ($q' \geq 0$) and $q''$ ($q'' \geq 0$) lattice points inside. Clearly, $p_4 - 2 + q' + q'' = q$. From the induction hypothesis, the area of the first smaller triangle is $\frac{p_1+p_3'+p_4-3}{2} + q' - 1$ and that of the second smaller triangle is $\frac{p_2+p_3''+p_4-3}{2} + q'' - 1$. The area of the original triangle, therefore, is $(\frac{p_1+p_3'+p_4-3}{2} + q' - 1) + (\frac{p_2+p_3''+p_4-3}{2} + q'' - 1)$, which equals $\frac{p_1+p_2+p_3'+p_3''}{2} + p_4 - 3 + q' - 1 + q'' - 1 = \frac{p_1+p_2+p_3+1}{2} + p_4 + q' + q'' - 5 = \frac{p_1+p_2+p_3-3}{2} + 2 + p_4 + q' + q'' - 5 = \frac{p}{2} + 2 + (p_4 - 2 + q' + q'') + 2 - 5 = \frac{p}{2} + q - 1$.

Case (b): Suppose the triangle has $p_1$, $p_2$, and $p_3$ lattice points on the three sides respectively and $q$ ($q > 0$) lattice points inside; so, $p_1 + p_2 + p_3 - 3 = p$. Select a lattice point that is inside the triangle and connect it to the three vertices with three line segments, to divide the triangle into three, every two of which share one newly drawn line segment as a side. The rest of the proof is similar to that of Case (a). □

3. Consider labeling the nodes of a full binary tree level by level, from top to bottom and left to right, with the numbers 1 through $n$, where $n$ is the number of nodes in the tree. Prove *by induction* that, for an internal node labeled $i$, its left and right children are labeled $2i$ and $2i + 1$ respectively.

   *Solution.* Let us count the levels of a full binary tree from 0, the root being on Level 0, its children on Level 1, etc.

   We claim and prove by induction that there are $2^l$ nodes on Level $l$ and the labeling as stated in the problem gives the $2^l$ nodes numbers $2^l$ through $2^{l+1} - 1$, for every $l$ such that $2^{l+1} - 1 \leq n$. This implies that, for an internal node labeled $i = 2^l + j$, $0 \leq j \leq 2^l - 1$, it is on Level $l$ and has $j$ siblings to the left, which totally have $2j$ children on Level $l + 1$ labeled $2^{l+1}$ through $2^{l+1} + 2j - 1$. So, the two children of the node labeled $i$ are labeled $(2^{l+1} + 2j - 1) + 1 = 2(2^l + j) = 2i$ and $(2^{l+1} + 2j - 1) + 2 = 2(2^l + j) + 1 = 2i + 1$ respectively, which is what the problem statement requires to be proved. Below is a proof of the claim by induction on the level $l$.

   Base case ($l = 0$): the root is the only node on Level 0 and is labeled $1 = 2^0$.

   Inductive step ($l > 0$ s.t. $2^{l+1} - 1 \leq n$): The nodes on Level $l$ are children of those on Level $l - 1$. From the induction hypothesis, there are $2^{l-1}$ nodes on Level $l - 1$. As each of the $2^{l-1}$ nodes has two children, there are $2^l$ nodes on Level $l$. Also, from the induction hypothesis, the $2^{l-1}$ nodes on Level $l - 1$ are labeled $2^{l-1}$ through $2^l - 1$. Therefore, the $2^l$ nodes on Level $l$ should be labeled $(2^l - 1) + 1$ through $(2^l - 1) + 2^l$, i.e., $2^l$ through $2^{l+1} - 1$. □

4. Consider the problem of merging two skylines, which is a useful building block for computing the skyline of a number of buildings. A skyline is an alternating sequence of $x$

coordinates and $y$ coordinates (heights), ending with an $x$ coordinate (as discussed in class). The sequence of coordinates may be conveniently stored in an array, say $A$, with $A[0]$ storing the first $x$ coordinate, $A[1]$ the first $y$ coordinate, $A[2]$ the second $x$ coordinate, etc.

Design a linear-time procedure that prints out the resulting skyline from merging two given skylines. Please present the procedure in suitable pseudocode. The procedure should be named `merge_skylines` and invoked by `merge_skylines(A,m,B,n)`, where `A` and `B` are the two input skylines and `A[m]` and `B[n]` store the final $x$ coordinate of skyline `A` and that of skyline `B` respectively. Does your procedure really run in $O(m+n)$ time? Please explain.

*Solution.*

```
merge_skylines(A,m,B,n)
// assume m,n >= 2.
begin
  if A[0] < B[0] then
    print A[0], A[1];
    merge_a(A[1], 0, A[2..m], m-2, B, n)
  else
    if A[0] > B[0] then
      print B[0], B[1];
      merge_b(0, B[1], A, m, B[2..n], n-2)
    else // A[0] = B[0]
      if A[1] < B[1] then
        print B[0], B[1];
        merge_b(A[1], B[1], A[2..m], m-2, B[2..n], n-2)
      else // A[1] > B[1] or A[1] = B[1] (given A[0] = B[0])
        print A[0], A[1];
        merge_a(A[1], B[1], A[2..m], m-2, B[2..n], n-2)
      end if
    end if
  end if
end


merge_a(ya, yb, A, m, B, n);
// ya, yb are the previous y coordinates of A and B, respectively.
// ya > yb.
begin
  if m = 0 and n = 0 then
    if A[0] < B[0] then
      print A[0], yb, B[0]
    else
      print A[0]
    end if;
    return
  end if;
  if m = 0 then
    if A[0] < B[0] then
```

```
    print A[0], yb, each entry of B
  else
    if ya >= B[1] then
      merge_a(ya, B[1], A, m, B[2..n], n-2)
    else
      print B[0], B[1];
      merge_b(ya, B[1], A, m, B[2..n], n-2)
  end if;
  return
end if;
if n = 0 then
  if A[0] < B[0] then
    if A[1] < yb then
      print A[0], yb;
      merge_b(A[1], yb, A[2..m], m-2, B, n)
    else
      print A[0], A[1];
      merge_a(A[1], yb, A[2..m], m-2, B, n)
    end if
  else // A[0] >= B[0]
    print each entry of A
  end if;
  return
end if;
// m,n >= 2
if A[0] < B[0] then
  if A[1] > yb then
    print A[0], A[1];
    merge_a(A[1], yb, A[2..m], m-2, B, n)
  else
    print A[0], yb;
    merge_b(A[1], yb, A[2..m], m-2, B, n)
  end if
else
  if A[0] > B[0] then
    if B[1] > ya then
      print B[0], B[1];
      merge_b(ya, B[1], A, m, B[2..n], n-2)
    else
      merge_a(ya, B[1], A, m, B[2..n], n-2)
    end if
  else // A[0] = B[0]
    if A[1] < B[1] then
      if B[1] != ya then
        print B[0], B[1];
      end if;
      merge_b(A[1], B[1], A[2..m], m-2, B[2..n], n-2)
    else // A[1] > B[1] or A[1] = B[1] (given A[0] = B[0])
      print A[0], A[1];
```

```
      merge_a(A[1], B[1], A[2..m], m-2, B[2..n], n-2)
    end if
  end if
 end if
end

merge_b(ya, yb, A, m, B, n);
// ya, yb are the previous y coordinates of A and B, respectively.
// ya < yb.
// analogous to merge_a.
```

□

5. Below is the pseudocode of the binary search algorithm we discussed in class. Would the code still be correct if we change the assignment "$Middle := \lceil \frac{Left+Right}{2} \rceil$" to "$Middle := \lfloor \frac{Left+Right}{2} \rfloor$" for $Middle$ to take instead the largest integer less than or equal to $\frac{Left+Right}{2}$? Please justify your answer.

**function Find** $(z, Left, Right)$ : *integer*;
**begin**
  **if** $Left = Right$ **then**
    **if** $X[Left] = z$ **then** $Find := Left$
    **else** $Find := 0$
  **else**
    $Middle := \lceil \frac{Left+Right}{2} \rceil$;
    **if** $z < X[Middle]$ **then**
      $Find := Find(z, Left, Middle - 1)$
    **else**
      $Find := Find(z, Middle, Right)$
**end**

**Algorithm Binary_Search** $(X, n, z)$;
**begin**
  $Position := Find(z, 1, n)$;
**end**

*Solution.* The code would be incorrect, if just that change is made. Consider $X[1..2] = [7, 9]$, an array with two numbers 7 and 9. Suppose we invoke **Binary_Search**$(X, 2, 6)$ to find out whether 6 is in $X$. The call in turns invokes $Find(6, 1, 2)$, whose execution will set $Middle$ to $\lfloor \frac{Left+Right}{2} \rfloor = \lfloor \frac{1+2}{2} \rfloor = 1$. Since $z = 6 < 7 = X[1] = X[Middle]$, the execution will invoke $Find(z, Left, Middle - 1)$, i.e., $Find(6, 1, 0)$, which will result in an access to $X[0]$, an erroreous behavior. □

6. Given the array below as input [to the Mergesort algorithm], what are the contents of array *TEMP* after the merge part is executed for the first time and what are the contents of *TEMP* when the algorithm terminates? Assume that each entry of *TEMP* has been initialized to 0 when the algorithm starts.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 9 | 2 | 6 | 5 | 10 | 8 | 3 | 1 | 12 | 4 | 11 |

*Solution.* The contents of array *TEMP* after the merge part is executed for the first time:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The contents of array *TEMP* when the algorithm terminates:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 0 | 0 |

□

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

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.

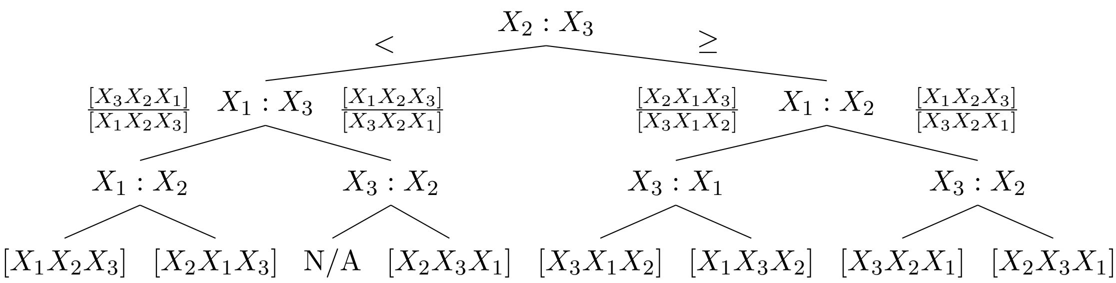
*Solution.*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 2 | 8 | 5 | 1 | 14 | 7 | 6 | 3 | 11 | 10 | 13 | 15 | 12 | 9 |
| 4 | 2 | 8 | 5 | 1 | 14 | $\underline{12}$ | 6 | 3 | 11 | 10 | 13 | 15 | $\underline{7}$ | 9 |
| 4 | 2 | 8 | 5 | 1 | $\underline{15}$ | 12 | 6 | 3 | 11 | 10 | 13 | $\underline{14}$ | 7 | 9 |
| 4 | 2 | 8 | 5 | $\underline{11}$ | 15 | 12 | 6 | 3 | $\underline{1}$ | 10 | 13 | 14 | 7 | 9 |
| 4 | 2 | 8 | $\underline{6}$ | 11 | 15 | 12 | $\underline{5}$ | 3 | 1 | 10 | 13 | 14 | 7 | 9 |
| 4 | 2 | $\underline{15}$ | 6 | 11 | $\underline{14}$ | 12 | 5 | 3 | 1 | 10 | 13 | $\underline{8}$ | 7 | 9 |
| 4 | $\underline{11}$ | 15 | 6 | $\underline{10}$ | 14 | 12 | 5 | 3 | 1 | $\underline{2}$ | 13 | 8 | 7 | 9 |
| $\underline{15}$ | 11 | $\underline{14}$ | 6 | 10 | $\underline{13}$ | 12 | 5 | 3 | 1 | 2 | $\underline{4}$ | 8 | 7 | 9 |

□

8. Draw a decision tree of the Heapsort algorithm (in increasing order) for the case of $A[1..3]$, i.e., $n = 3$. In the decision tree, you must indicate (1) which two elements of the original input array are compared in each internal node and (2) the sorting result in each leaf. Please use $X_1$, $X_2$, $X_3$ (not $A[1]$, $A[2]$, $A[3]$) to refer to the elements (in this order) of the original input array $A$.

*Solution.*




Note: two or more of $X_1$, $X_2$, and $X_3$ may be equal. □

9. The *next* table is a precomputed table (for $B = b_1b_2 \cdots b_m$) that plays a critical role in the KMP algorithm. Under what condition regarding $b_1b_2 \cdots b_i$, $2 \leq i \leq m$, will $next[i]$ get a 0 in the preprocessing? And under what condition can it be safely set to $-1$ (without missing a potential match when searching for $B$ in another input string)?

*Solution.* The value of $next[i]$ is determined by the length of the longest prefix of $b_1 b_2 \cdots b_{i-1}$ that is also a suffix of $b_1 b_2 \cdots b_{i-1}$. When no such prefix exists, $next[i]$ gets a 0.

During a search for string $B$ in string $A$ using KMP, when $b_j$ is compared against $a_i$ and the comparison fails, $b_{next[j]+1}$ is tried next against $a_i$. When $next[j] = 0$, it is $b_1$ that is compared with $a_i$. If the comparison fails, then $b_1$ will be compared against $a_{i+1}$, according to the case for $next[j]+1 = 0$, i.e., $next[j] = -1$. When $b_1 = b_j$, the comparison between $b_1$ and $a_i$ is doomed to fail (since $b_1 = b_j \neq a_i$) and the comparison could have been saved. To achieve the saving, we can set $next[j]$ to $-1$ (instead of 0) when $b_j$ happens to be equal to $b_1$. □

10. Given two strings $A = bbaaba$ and $B = ababa$, what is the result of the minimal cost matrix $C[0..6, 0..5]$, according to the algorithm discussed in class for changing A character by character into B? Aside from giving the cost matrix, please show the details of how the entry $C[4,3]$ is computed from the values of $C[3,2]$, $C[3,3]$, and $C[4,2]$.

    *Solution.*

| | | $a$ | $b$ | $a$ | $b$ | $a$ |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| $b$ | 1 | 1 | 1 | 2 | 3 | 4 |
| $b$ | 2 | 2 | 1 | 2 | 2 | 3 |
| $a$ | 3 | 2 | 2 | 1 | 2 | 2 |
| $a$ | 4 | 3 | 3 | 2 | 2 | 2 |
| $b$ | 5 | 4 | 3 | 3 | 2 | 3 |
| $a$ | 6 | 5 | 4 | 3 | 3 | 2 |

$$C[4,3] = \min \left\{ \begin{array}{ll} C[3,3]+1 = 1+1 = 2 & \text{(deleting } A_4\text{)}, \\ C[4,2]+1 = 3+1 = 4 & \text{(inserting } B_3\text{)}, \\ C[3,2] = 2 & (A_4 = B_3) \end{array} \right\} = 2$$

□

## Appendix

- The Mergesort algorithm:

**Algorithm Mergesort** $(X, n)$;
**begin** $M\_Sort(1, n)$ **end**

**procedure M_Sort** $(Left, Right)$;
**begin**
    **if** $Right - Left = 1$ **then**
      **if** $X[Left] > X[Right]$ **then** $swap(X[Left], X[Right])$
    **else if** $Left \neq Right$ **then**
        $Middle := \lceil \frac{1}{2}(Left + Right) \rceil$;
        $M\_Sort(Left, Middle - 1)$;
        $M\_Sort(Middle, Right)$;
        // the merge part
        $i := Left$; $j := Middle$; $k := 0$;
        **while** $(i \leq Middle - 1)$ and $(j \leq Right)$ **do**
            $k := k + 1$;

```
            if X[i] ≤ X[j] then
                TEMP[k] := X[i];  i := i + 1
            else TEMP[k] := X[j];  j := j + 1;
      if j > Right then
         for t := 0 to Middle − 1 − i do
            X[Right − t] := X[Middle − 1 − t]
      for t := 0 to k − 1 do
         X[Left + t] := TEMP[1 + t]
end
```

- The KMP algorithm (assuming *next*):

```
Algorithm String_Match (A, n, B, m);
begin
    j := 1;  i := 1;
    Start := 0;
    while Start = 0 and i ≤ n do
        if B[j] = A[i] then
            j := j + 1;  i := i + 1
        else
            j := next[j] + 1;
            if j = 0 then
                j := 1;  i := i + 1;
        if j = m + 1 then Start := i − m
end
```

- The algorithm for computing the *next* table in the KMP algorithm:

```
Algorithm Compute_Next (B, m);
begin
    next[1] := −1;  next[2] := 0;
    for i := 3 to m do
        j := next[i − 1] + 1;
        while B[i − 1] ≠ B[j] and j > 0 do
            j := next[j] + 1;
        next[i] := j
end
```