AALBORG UNIVERSITY

Department of Electronic Systems

Algorithmer

2023 Solution to Workshop problems

Semester III Academic Year 2023-2024

For programmes: **Computerteknologi (COMTEK), Electronik og systemdesign (ESD) and Cyber- og computer-teknologi (CCT)**

Date: 05/12/2023 **Time and Duration:** 180 Minutes

Weighting: 0 % **No. pages:** 11 **No. sections:** 10 **Total marks:** 00

Comments about the solutions:

For Lecturer Use Only

Section Number	Total Marks	Score
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
Total		

1. This solution set is provided to help you understand and assess your answers to the workshop problems.
2. Efforts have been made to organize the solutions in a way that minimizes the need for repeated drawing thereby avoiding wasted time.
3. If you have doubts about any of the provided answers, please reach out to me.

Problem 1: (Question One) [0 points]

(a) (2 points) Which of the following statements are true about the function $T(n) = 10n^2 + 5n(n - \log_2(n))$. If there are multiple correct statements, mark all.

- ◯ $T(n) \in O(n^2)$ and $T(n) \in \Theta(n^2)$.
- ◯ $T(n) \in O(n \log_2(n))$ and $T(n) \in \Theta(n^2)$.
- √ $T(n) \in O(n^2)$ **and** $T(n) \in \Theta(n \log_2(n))$**.**

(b) (4 points) Suppose that for inputs of size n on a particular machine, algorithms A and B runs in $8n^2$ and $64n \log_2(n)$ steps. For what values of n is Algorithm A more efficient than Algorithm B? *You must Show how you arrive at your answer.*

Solution: We wish to establish the values of n for wish $8n^2 < 64n \log_2(n)$. This gives $n < 8 \log_2(n)$ or $n \leq 43$.
Algorithm A is therefore more efficient than B for values of n less than or equal to 43.

(c) (4 points) A computer engineering student claims that the function $8n \log_2(n) + 4n$ is $O(n \log_2(n))$. Show that the claim is correct and provide at least one pair of witnesses to support the claim.

Solution: Claim: $8n \log_2(n) + 4n \in O(n \log_2(n))$
Proof: A function $f(n)$ is $O(n)$ iif $0 \leq f(n) \leq cg(n) \qquad \forall n \geq n_0$
To prove the student's claim, we must show that

$$8n \log_2(n) + 4n \leq cn \log_2(n)$$

Dividing the above equation by $n \log_2(n)$ yields:

$$8 + \frac{4}{\log_2(n)} \leq c$$

By selecting $n_0 = 2$, we have from the above equation that: $n \log_2(n)$ yields:

$$8 + \frac{4}{\log_2(2)} \leq c \rightarrow 12 \leq c$$

The above shows that we can find values of c and n_0 for which $8n \log_2(n) + 4n \leq cn \log_2(n)$. The student is therefore correct and a witness is $(c = 12, n_0 = 2)$.

Problem 2: (Question Two) [0 points]

(a) (2 points) What is the solution to the recurrence relation $T(n) = 4T(n/2) + n^3$?

$\surd$ $T(n) \in \Theta(n^3)$ $\bigcirc$ $T(n) \in \Theta(n)$ $\bigcirc$ $T(n) \in \Theta(n^2 \log_2(n))$ $\bigcirc$ $T(n) \in \Theta(n \log_2(n))$

(b) (8 points) Let $A[1 : n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, the pair (i, j) is called an inversion of A. Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \log_2(n))$ worst-case time. (*Hint: Modify the merge sort algorithm.*)

Solution: We can modify the merge-sort algorithm to count the number of inversions as shown in the algorithm below.

```
1: procedure INVERSIONS-COUNT(A, p, r)
2:     if p < r then
3:         q = ⌊((p + r)/2)⌋
4:         left = INVERSIONS − COUNT(A, p, q)
5:         right = INVERSIONS − COUNT(A, q + 1, r)
6:         numInversions = MERGE − INVERSIONS(A, p, q, r) + left + right
7:         return numInversions
8:     end if
9: end procedure
```

The algorithm requires a modified version of the MERGE algorithm for merge-sort. This modified algorithm named MERGE-INVERSIONS is used on line (6) of INVERSION - COUNT algorithm above. The pseudocode for MERGE-INVERSIONS is as follows:

```
procedure MERGE-INVERSIONS(A, p, q, r)
    n1 = q − p + 1
    n2 = r − q
    Let L[1, ··· , n1 + 1] and R[1, ··· , n2 + 1] be new arrays
    for i = 1 to n1 do
        L[i] = A[p + i − 1]
    end for
    for j = 1 to n2 do
        R[j] = A[q + j]
    end for
    L[n1 + 1] = ∞
    R[n2 + 1] = ∞
    i = 1
    j = 1
    inversions = 0
    for k = p to r do
        if L[i] ≤ R[j] then
            A[k] = L[i]
            i = i + 1
        else
```

$inversions = inversions + n_1 - i + 1$
$j = j + 1$
end if
end for
return *inversions*
end procedure

Similar to MERGE-SORT, algorithm INVERSION-COUNT runs in worst-case time of $\Theta(n \log_2(n))$.

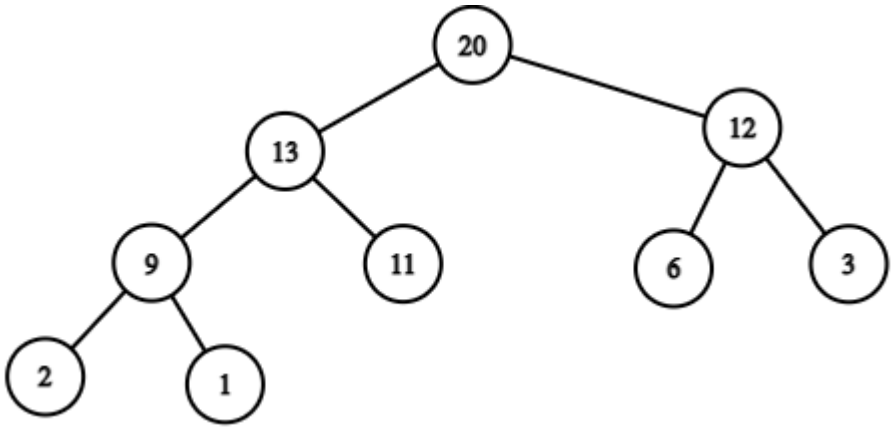
Problem 3: (Question Three) [0 points]

(a) (1 point) Is the array with values $\{1, 2, 4, 5, 6, 8\}$ a min-heap?

- √ **Yes**
- ○ No

(b) (3 points) Sketch the binary tree representation of a max-heap with the following numbers: $\{9, 13, 20, 2, 3, 6, 12, 11, 1\}$. You may use the numbers in any order.

Solution: Array representation of a possible max-heap: $\{20, 13, 12, 9, 11, 6, 3, 2, 1\}$ The binary tree representation is thus:



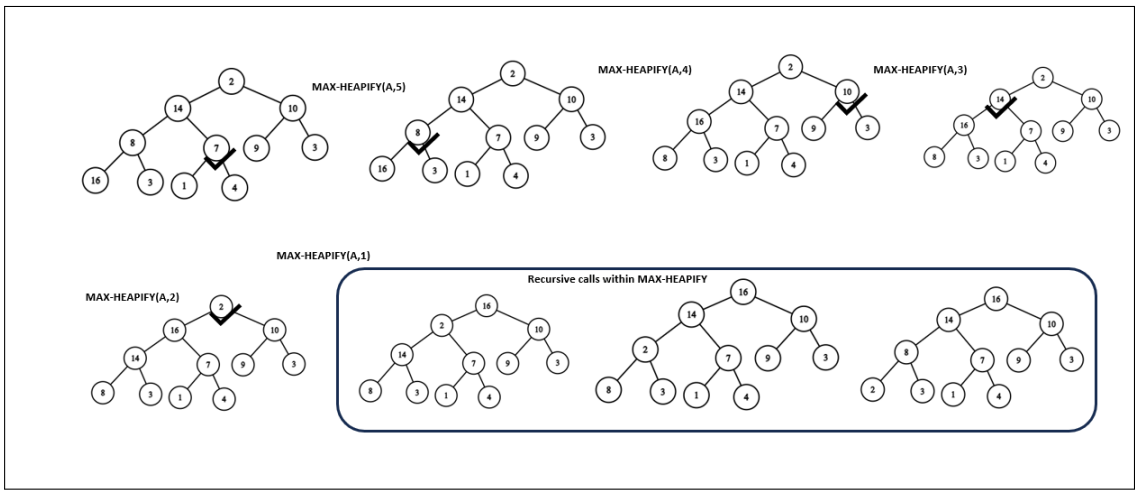
Note: Other binary trees fulfilling the max-heap property are possible.

(c) (6 points) Illustrate with sketches the operations of the BUILD-MAX-HEAP algorithm using the heap represented by the array $A = [2, 14, 10, 8, 7, 9, 3, 16, 4, 1]$

Solution: According to the BUILD-MAX-HEAP algorithm, the required steps are:

- The algorithm starts at $i = 5$
- Max-heapify is applied on subtrees rooted at nodes labeled 7, 8, 10, 14, 2.

The operation of BUILD-MAX-HEAP is illustrated in the figure below where the node with a tick indicates the one on which max-heapify is called.


MAX-HEAPIFY(A,5)
MAX-HEAPIFY(A,4)
MAX-HEAPIFY(A,3)
MAX-HEAPIFY(A,2)
MAX-HEAPIFY(A,1)
Recursive calls within MAX-HEAPIFY

Problem 4: (Question Four) [0 points]

The Fibonacci numbers are defined by the recurrence:

$$F_n = \begin{cases} 0; & i = 0, \\ 1; & n = 1, \\ F_{n-1} + F_{n-2}; & n \geq 2 \end{cases}$$

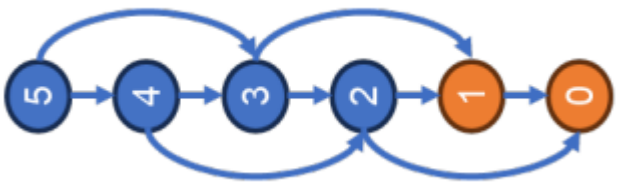
(a) (4 points) Give a linear-time dynamic-programming algorithm to compute the nth Fibonacci number.

Solution: A bottom-up dynamic programming algorithm for computing the nth Fibonacci number is given below:

```
procedure DYNAMIC-FIB(n)
    F_prev = 1
    F_pprev = 1
    if n ≤ 1 then
        return 1
    end if
    for i = 2 to n do
        F_temp = F_prev + F_pprev
        F_pprev = F_prev
        F_prev = F_temp
    end for
    return F_prev
end procedure
```

(b) (3 points) Draw the subproblem graph with $n = 5$.

Solution: Sub-problem graph for $n = 5$:



(c) (3 points) Give expressions for the number of vertices and edges in the subproblem graph in (b) as a function of n.

Solution: Number of vertices and edges in the subproblem graph as a function of n: There are $n + 1$ vertices corresponding to subproblems $FIB(0), FIB(1), \cdots, FIB(n)$. Each vertex from 2 to n has two outgoing edges and nodes 0 and 1 has none. The

number of edges is therefore

$$2 \times (n-1) = 2n - 2$$

Ans: There are $n + 1$ vertices and $2n - 2$ edges in the subproblem graph.

Problem 5: (Question Five) [0 points]

(a) (3 points) Suppose that we have numbers between 1 and 600 in a binary search tree, and we want to search for the number 363. Which of the following could NOT be the sequence of nodes examined?

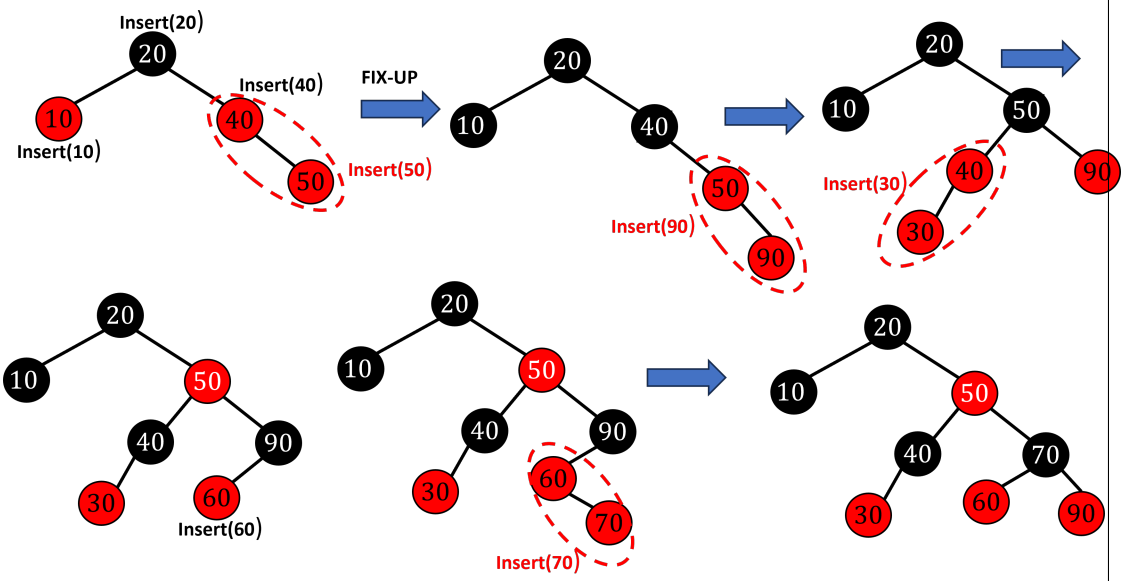
- √ 535, 278, 347, 421, 299, 392, 358, 363**.**
- ◯ 2, 399, 387, 219, 266, 382, 381, 278, 363.
- √ 525, 202, 511, 240, 512, 245, 363**.**
- ◯ 2, 252, 401, 398, 330, 344, 397, 363.
- ◯ 524, 220, 511, 244, 498, 258, 362, 363.

(b) (7 points) Sketch the trace of inserting the following numbers 20, 10, 40, 50, 90, 30, 60, 70 into an initially empty Red-Black Tree. Determine the RB height of the result tree.
Note: In a Red-Black Tree, every new node must be inserted with the color RED and after every insertion operation, we must check all the properties of Red-Black Tree. If no violations are discovered, we move to the next operation, otherwise, we perform the operations ***rotation*** *and/or* ***recolor****.*

Solution: The steps of inserting the numbers into a red-black tree are listed below:

1. Insert(20): RB-tree is empty. Insert node with key = 20 as the root and color it Black.
2. Insert(10): Insert node with key 10 in the right subtree of 10 and color it red.
3. Insert(40): Insert node with key 40 in the left subtree of 10 and color it red.
4. Insert(50): Insert node with key 40 in the left subtree of 40 and color it red. Now we have a violation, i.e., 2 successive red nodes. Apply recolor and/or rotation.
5. Insert(90): Insert node with key 90 as the right child of 50 and color it red. Again we have a violation, i.e., 2 successive red nodes. Apply recolor and/or rotation.
6. Continue the process until all nodes have been inserted.

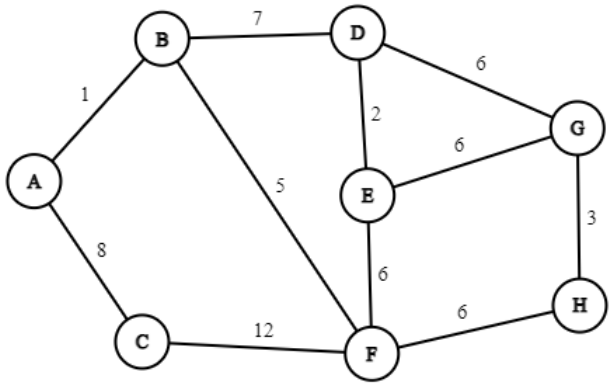
A sketch of the insertion steps is given below. The blue arrow below trees indicates the point where RB-INSERT-FIX-UP is called to correct a violation of a property. NIL nodes are omitted for simplicity.



The **black-height** of the resulting tree is 2.

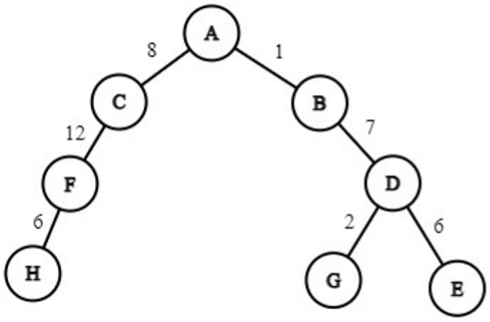
Problem 6: (Question Six) [0 points]

Consider the following undirected graph, G on 8 vertices labeled A–H.

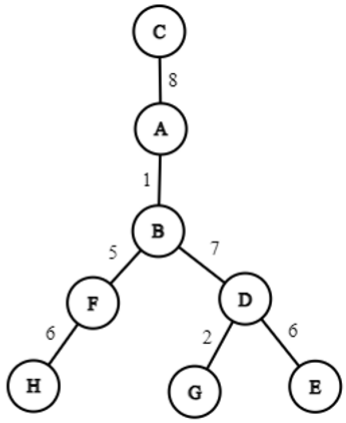


(a) (2 points) Sketch at least 2 spanning trees for G and compute the weight $w(T)$ for each of the sketched spanning trees.

Solution: Two spanning trees with $w(T) = 42$ and $w(T) = 33$ are shown below:



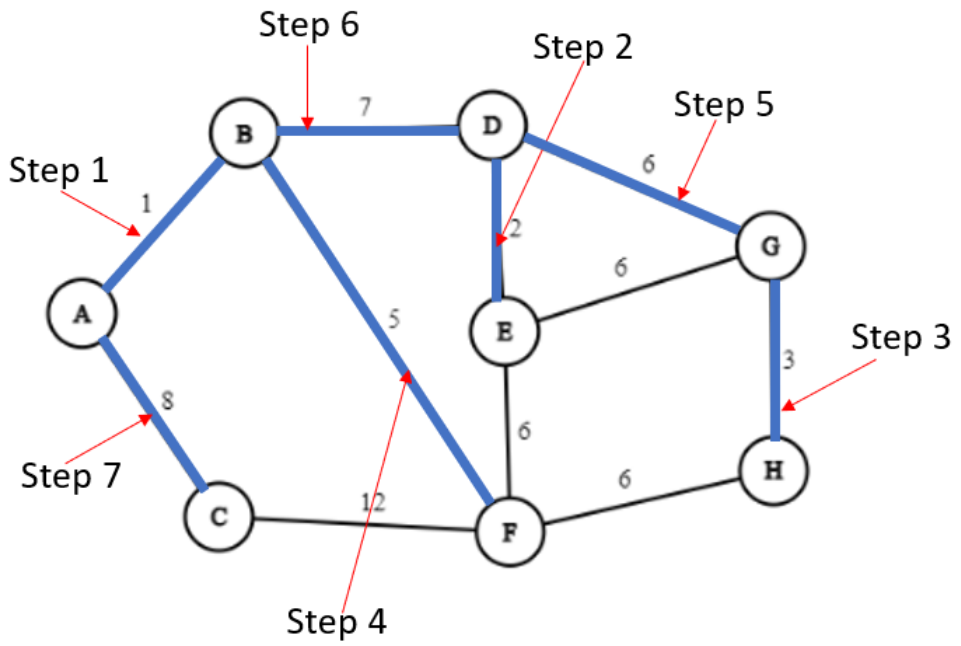
$w(T)) = 8 + 1 + 12 + 7 + 6 + 2 + 6 = 42$



$w(T) = 8 + 1 + 5 + 7 + 6 + 2 + 6 = 33$

(b) (8 points) Give a trace of the steps for computing a spanning tree of minimal weight for G using Kruskal's algorithm.

Solution: The Kruskal's algorithm considers each edge in the graph in a sorted order. The trace of the execution of the algorithm on the above graph is given below. The edge added to the forest at each step is colored blue. An edge is added if and only if it joins two distinct trees in the forest.

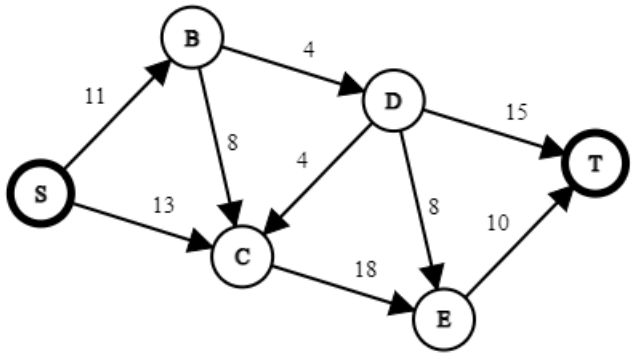


The resulting minimum spanning tree has a weight

$$w(T) = 1 + 2 + 3 + 5 + 6 + 7 + 8 = 32$$

Problem 7: (Question Seven) [0 points]

Consider the following flow network, G with the source and sink nodes denoted S and T, respectively.

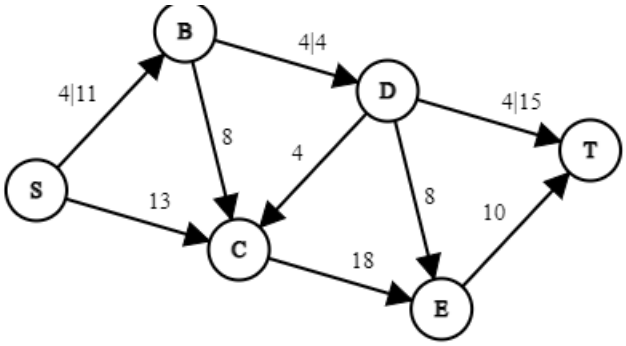


(a) (7 points) Show the execution of the Ford-Fulkerson algorithm on the flow network above.

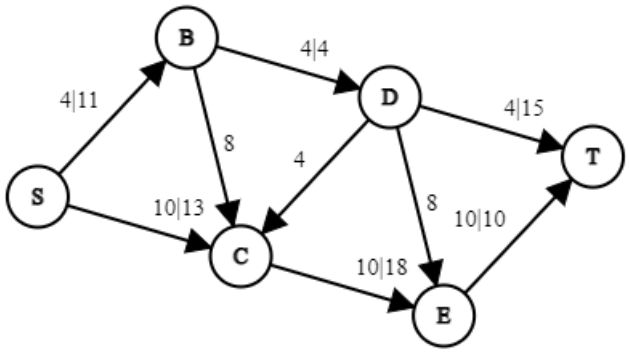
Solution: The Ford-Fulkerson algorithm starts with the given graph, G as the residual network, G_f, and perform the following steps:

- Find an augmenting path in G_f
- Augment the flow along the augmenting path in G
- Construct a residual network using the augmented network.
- Repeat the above steps until no augmenting path can be found in the residual network.

An augmenting path in the initial flow network is $S \rightarrow B \rightarrow D \rightarrow T$ with $c_f(p) = 4$. The augmented flow network is then:



From G_f for the updated network above, an augmenting path $S \rightarrow C \rightarrow E \rightarrow T$ with $c_f(p) = 10$. Augmenting the flow network yield:



The residual network for the augmented network above has no $S \rightarrow T$ path (augmenting path) and is, therefore, the network with maximum flow.

(b) (1 point) What is the maximal flow in the resulting network from (a) above?

Solution: The maximal is the flow value in the maximum flow network:

$$|f*| = f(S,B) + f(S,C) = 14$$

(c) (2 points) What do you need to change in the resulting network from (a) to violate the capacity constraint?

Solution: The capacity constraint requires that the flow on an edge is at most equal to its capacity. This can be violated by making the flow larger than the capacity for any of the edges, e.g., setting $f(S,B) > 11$ or $f(B,D) > 4$.

Problem 8: (Question Eight) [0 points]

(a) (2 points) A *collision* in a hash table occurs when

- ◯ Two key-value pairs with equal keys have different values
- √ **Two key-value pairs with different keys generate the same hash value**
- ◯ Two key-value pairs with different keys generate different hash values
- ◯ Two key-value pairs with equal keys generate different hash values

(b) (5 points) A quadratic-probing hash table of length, $m = 11$ uses the hash function $h(k, i) = (k + c_1 i + c_2 i^2) \mod 11$ with $c_1 = 1$ and $c_2 = 3$. After inserting nine integer keys into an initially empty hash table, the result is:

Slot index	0	1	2	3	4	5	6	7	8	9	10
key	22		88	17	4		28	59	15	31	10

Determine an appropriate insertion sequence resulting in the hash table above. Show how you arrive at your answer.

Solution: The insertion sequence can be determined by determining the hash value of each key at successive probe index, i as shown below. The slot occupied by each key in the result is colored red.

Key/Probe index	0	1	2	3	4	5	6	7	8	9
22	0	4	3	8	8	3	4	0	2	10
88	0	4	3	8	8	3	4	0	2	10
17	6	10	9	3	3	9	10	6	8	5
4	4	8	7	1	1	7	8	4	6	3
28	6	10	9	3	3	9	10	6	8	5
59	4	8	7	1	1	7	8	4	6	3
15	4	8	7	1	1	7	8	4	6	3
31	9	2	1	6	6	1	2	9	0	8
10	10	3	2	7	7	2	3	10	1	9

Based on the table above, a possible insertion sequence is:

$$22 \rightarrow 4 \rightarrow 31 \rightarrow 10 \rightarrow 28 \rightarrow 15 \rightarrow 59 \rightarrow 17 \rightarrow 88$$

(c) (3 points) Give the resulting hash table if the keys are inserted into the initially empty table with linear probing (with $h(k, i) = (k + i) \mod 11$.

Solution: Applying linear probing using the ordering in the answer from (b) gives:

Slot index	0	1	2	3	4	5	6	7	8	9	10
key	22	88			4	15	28	59	17	31	10

Alternative correct answer using the order in the given table:

Slot index	0	1	2	3	4	5	6	7	8	9	10
key	22	88			4	59	17	28	15	31	10

Problem 9: (Question Nine) [0 points]

Consider the following algorithm that sorts an integer array with length N.

Algorithm 1 Sorts an integer array, A.

```
1:  procedure sortingAlgorithm(A, N)
2:      for i = 1 to N do
3:          for j = N down to i + 1 do
4:              if A[j] < A[j − 1] then
5:                  temp = A[j]
6:                  A[j] = A[j − 1]
7:                  A[j − 1] = temp
8:              end if
9:          end for
10:     end for
11:     Return A
12: end procedure
```

(a) (2 points) Which of the following is true about the sorting algorithm above? Check all that apply.

- √ **The sorting algorithm is stable.**
- √ **The sorting algorithm is in place.**

(b) (6 points) Using appropriate notations, determine the *best case* and *worst case* complexities of the algorithm in terms of the number of *comparisons*.

Solution: The algorithm performs the same amount of comparisons regardless of the actual form of the input array A.

The algorithm performs $N - i$ comparisons for each $i = 1, 2, \cdots, N$.

$$T(n) = \sum_{i=1}^{N}(N - i) = N^2 - \sum_{i=1}^{N}(N - i) = N^2 - \frac{N(N+1)}{2} \in O(N^2)$$

(c) (2 points) How does the algorithm above compare with insertion sort in terms of the number of exchanges?

Solution: Insertion sort makes the same number of exchanges a SORTINGALGORITHM for all arrays

Problem 10: (Question Ten) [0 points]

An array B is a *circular shift* of $A = [A_1, A_2, \cdots, A_N]$ if it consists of the subarray $A_{p+1}, A_{p+2}, \cdots, A_N$ followed by $A_1, A_2, \cdots, A_p$ for some integer p. For example, the array $B = [10, 12, 15, 2, 5, 7, 8, 9]$ is a circular shift of the array $A = [2, 5, 7, 8, 9, 10, 12, 15]$ with $p = 5$ and $N = 8$.

(a) (6 points) Suppose that you are given an array B that is a circular shift of some sorted array, clearly specify the pseudocode for an efficient algorithm to determine whether a given integer key appears in the array B. *The complexity of your algorithm must be at most logarithmic in the worst case.*

Solution: This problem can be solved using the *divide-and-conquer* algorithm - binary search based on the following observation.

If a circular shift, B of a sorted array, A is split into two halves at any of its indices, one of the two halves is sorted.

```
1:  procedure searchShift(A, N, key)
2:      low = 1
3:      high = N
4:      while low ≤ high do
5:          mid = ⌊(low + high)/2⌋
6:          if A[mid] == key then
7:              return mid
8:          end if
9:          if A[low] ≤ A[mid] then
10:             if A[low] <= key and key <= A[mid] then
11:                 high = mid
12:             else
13:                 low = mid + 1
14:             end if
15:         else
16:             if A[mid + 1] <= key and key <= arr[high] then
17:                 low = mid + 1
18:             else
19:                 high = mid
20:             end if
21:         end if
22:     end while
23:     Return NotFound
24: end procedure
```

(b) (1 point) Which algorithmic design principle is used in your algorithm from (a) above?

Solution: The algorithm uses the divide and conquer principle.

(c) (3 points) Write an expression for the exact and asymptotic worst-case complexity of your algorithm.

Solution: The analysis of the algorithm in (b) is the same as that of binary search: we successively split the input into two parts, eliminate one part, and continue on the other part until the key is found or the array can no longer be split.

The complexity can be expressed as the recurrence:

$$T(N) = T(N/2) + O(1)$$

Solving the recurrence gives the worst-case complexity as

$$T(n) = O(\log_2(N)$$