Appendix

Answers to the Questions

Chapter 2: Introduction to Algorithm Design

Question 1

Find the time complexity of the following Python snippets:

a.

```
i=1
while(i<n):
    i*=2
    print("data")
```

b.

```
i =n
while(i>0):
    print("complexity")
    i/ = 2
```

c.

```
for i in range(1,n):
    j = i
    while(j<n):
        j*=2
```

d.

```
i=1
while(i<n):
    print("python")
    i = i**2
```

Solution

a. The complexity will be `O(log(n))`.

   As we are multiplying the integer `i` by `2` in each step there will be exactly `log(n)` steps. (`1`, `2`, `4`, ... till `n`).

b. The complexity will be `O(log(n))`.

   As we are dividing the integer `i` by `2` in each step there will be exactly `log(n)` steps. (`n`, `n/2`, `n/4`, ... till `1`).

c. The outer loop will run `n` times for each `i` in the outer loop, while the inner `while` loop will run `log(i)` times because we are multiplying each of the `j` values by `2` until it is less than `n`. Hence, there will be a maximum of `log(n)` steps in the inner loop. Therefore, the overall complexity will be `O(nlog(n))`.

   In this code snippet, the `while` loop will execute based on the value of `i` until the condition becomes `false`. The value of `i` is incrementing in the following series:

   ```
   2, 4, 16, 256, ... n
   ```

   We can see that the number of times the loop is executing is $\log_2(\log_2(n))$ for a given value of `n`. So, for this series there will be exactly $\log_2(\log_2(n))$ executions of the loop. Hence the time complexity will be $O(\log_2(\log_2(n))$.

Chapter 3: Algorithm Design Techniques and Strategies

Question 1

Which of the following options will be correct when a top-down approach of dynamic programming is applied to solve a given problem related to the space and time complexity?

a. It will increase both time and space complexity
b. It will increase the time complexity, and decrease the space complexity
c. It will increase the space complexity, and decrease the time complexity
d. It will decrease both time and space complexities

Solution

Option c is correct.

Since the top-down approach of dynamic programming uses the memoization technique, which stores the pre-calculated solution of a subproblem. It avoids recalculation of the same subproblem that decreases the time complexity, but at the same time, the space complexity will increase because of storing the extra solutions of the subproblems.

Question 2

What will be the sequence of nodes in the following edge-weighted directed graph using the greedy approach (assume node **A** as the source)?

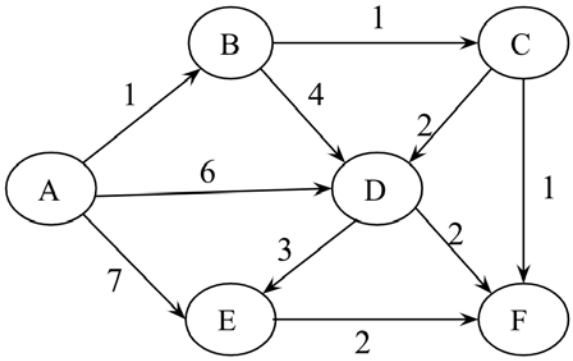

Figure A.1: A weighted directed graph

Solution

A, B, C, F, E, D

In Dijkstra's algorithm, at each, point we choose the smallest weight edge, which starts from any one of the vertices in the shortest path found so far, and add it to the shortest path.

Question 3

Consider the weights and values of the items in *Table 3.8*. Note that there is only one unit of each item.

Item	Weight	Value
A	2	10
B	10	8
C	4	5
D	7	6

Table A.1: The weights and values of different items

We need to maximize the value; the maximum weight should be 11 kg. No item may be split. Establish the values of the items using a greedy approach.

Solution

Firstly, we picked item A (weight 2 kg) as the value is the maximum (10). The second highest value is for item B, but as the total weight becomes 12 kg, this violates the given condition, so we cannot pick it. The next highest value is item D, and now the total weight becomes 2+7 = 9 kg (item A + item D). The next remaining item, C, cannot be picked because after adding it, the total weight condition will be violated.

So, the total value of the items picked up using the greedy approach = 10 + 6 = 16

Chapter 4: Linked Lists

Question 1

What will be the time complexity when inserting a data element after an element that is being pointed to by a pointer in a linked list?

Solution

It will be `O(1)`, since there is no need to traverse the list to reach the desired location where a new element is to be added. A pointer is pointing to the current location, and a new element can be directly added by linking it.

Question 2

What will be the time complexity when ascertaining the length of the given linked list?

Solution

`O(n)`.

In order to find out the length, each node of the list has to be traversed, which will take `O(n)`.

Question 3

What will be the worst-case time complexity for searching a given element in a singly linked list of length n?

Solution

`O(n)`.

In the worst case, the data element to be searched will be at the end of the list, or will not be present in the list. In that case, there will be a total `n` number of comparisons, thus making the worst-case time complexity `O(n)`.

Question 4

For a given linked list, assuming it has only one `head` pointer that points to the starting point of the list, what will be the time complexity for the following operations?

a. Insertion at the front of the linked list
b. Insertion at the end of the linked list
c. Deletion of the front node of the linked list
d. Deletion of the last node of the linked list

Solution

a. `O(1)`. This operation can be performed directly through the head node.
b. `O(n)`. It will require traversing the list to reach the end of the list.
c. `O(1)`. This operation can be performed directly through the head node.
d. `O(n)`. It will require traversing the list to reach the end of the list.

Question 5

Find the n^{th} node from the end of a linked list.

Solution

In order to find out the n^{th} node from the end of the linked list, we can use two pointers – `first` and `second`. Firstly, move the second pointer to `n` nodes from the starting point. Then, move both the pointers one step at a time until the second pointer reaches the end of the list. At that time, the first pointer will point to the n^{th} node from the end of the list.

Question 6

How can you establish whether there is a loop (or circle) in a given linked list?

Solution

To find out the loop in a linked list, it is most efficient to use **Floyd's cycle-finding algorithm**. In this approach, two pointers are used to detect the loop – let's say the first and second pointers. We start moving both the pointers from the starting point of the list.

We move the first and second pointers by one and two nodes at a time. If these two pointers meet at the same node, that indicates that there is a loop, otherwise, there is no loop in the given linked list.

The process is shown in the below figure with an example:

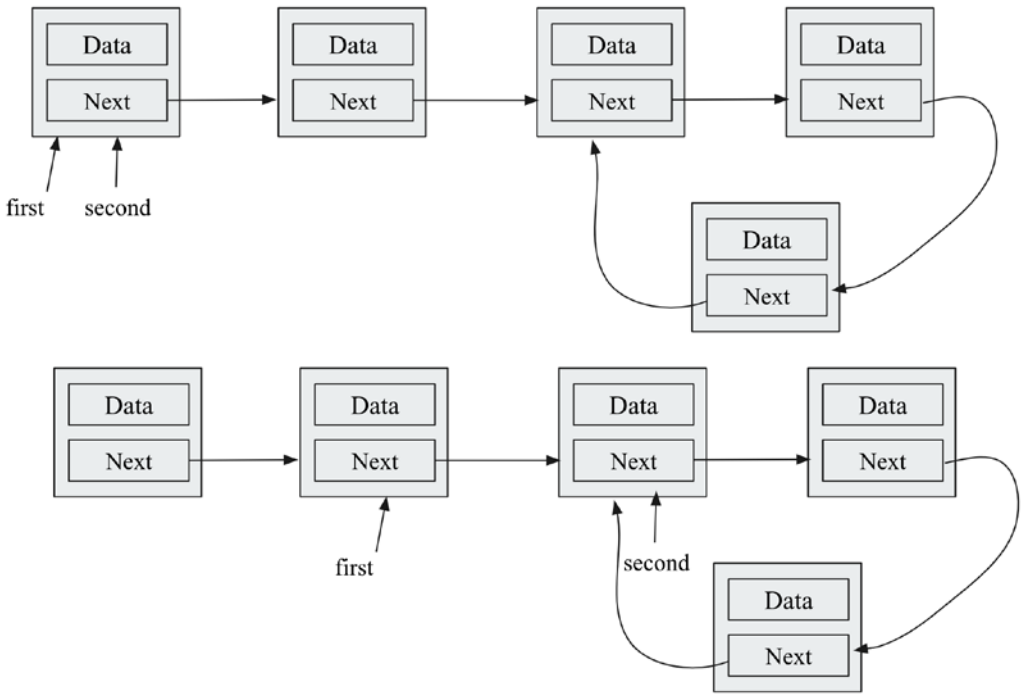

Figure A.2: Loop in a singly linked list

Question 7

How can you ascertain the middle element of the linked list?

Solution

It can be done with two pointers, say, the first and second pointers. Start moving these two pointers from the starting node. The first and second pointers should move one and two nodes at a time, respectively. When the second node reaches the end of the list, the first node will point to the middle element of the singly linked list.

Chapter 5: Stacks and Queues

Question 1

Which of the following options is a true queue implementation using linked lists?

a. If, in the enqueue operation, new data elements are added at the start of the list, then the dequeue operation must be performed from the end.

b. If, in the enqueue operation, new data elements are added to the end of the list, then the enqueue operation must be performed from the start of the list.

c. Both of the above.

d. None of the above.

Solution

B is correct. The queue data structure follows a FIFO order, hence data elements must be added to the end of the list, and then removed from the front.

Question 2

Assume a queue is implemented using a singly linked list that has `head` and `tail` pointers. The enqueue operation is implemented at `head`, and the dequeue operation is implemented at the `tail` of the queue. What will be the time complexity of the enqueue and dequeue operations?

Solution

The time complexity of the enqueue operation will be `O(1)` and `O(n)` for the dequeue operation. As for the enqueue operation, we only need to delete the `head` node, which can be achieved in `O(1)` for a singly linked list. For the dequeue operation, to delete the `tail`, we need to traverse the whole list first to the `tail`, and then we can delete it. For this we need linear, `O(n)`, time.

Question 3

What is the minimum number of stacks required to implement a queue?

Solution

Two stacks.

Using two stacks and the enqueue operation, the new element is entered at the top of `stack1`. In the dequeue process, if `stack2` is empty, all the elements are moved to `stack2`, and finally, the top of `stack2` is returned.

Question 4

The enqueue and dequeue operations in a queue are implemented efficiently using an array. What will be the time complexity for both of these operations?

Solution

`O(1)` for both operations.

If we use a circular array for the implementation of a queue, then we do not need to shift the elements, just the pointers, so we can implement both the enqueue and dequeue operations in `O(1)` time.

Question 5

How can we print the data elements of a queue data structure in reverse order?

Solution

Make an empty stack, then enqueue each of the elements from the queue and push them into the stack. After the queue is empty, start popping out the elements from the stack and then printing them one by one.

Chapter 6: Trees

Question 1

Which of the following is true about binary trees:

a. Every binary tree is either complete or full
b. Every complete binary tree is also a full binary tree
c. Every full binary tree is also a complete binary tree
d. No binary tree is both complete and full
e. None of the above

Solution

Option A is incorrect since it is not compulsory that a binary tree should be complete or full.

Option B is incorrect since a complete binary tree can have some nodes that are not filled in the last level, so a complete binary tree will not always be a full binary tree.

Option C is incorrect, as it is not always true, the following figure is a full binary tree, but not a complete binary tree:

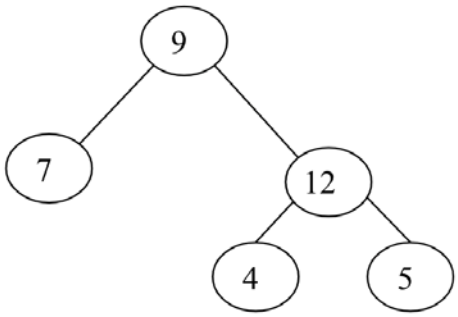

Figure A.3: A binary tree that is full, but not complete

Option D is incorrect, as it is not always true. The following tree is both a complete and full binary tree:

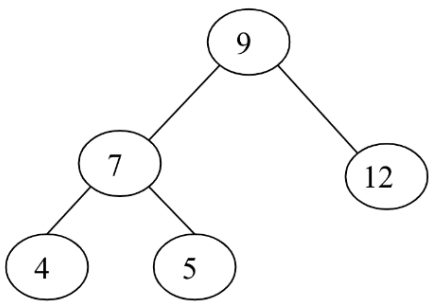

Figure A.4: A binary tree, that is full and complete

Question 2

Which of the tree traversal algorithms visit the root node last?

Solution

`postorder` traversal.

Using `postorder` traversal, we first visit the left subtree, then the right subtree, and finally we visit the `root` node.

Question 3

Consider this binary search tree:

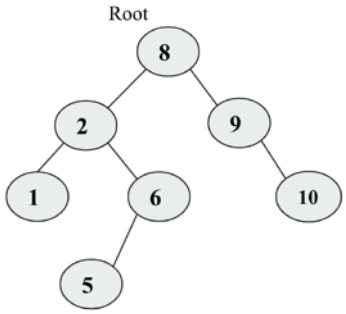

Figure A.5: Sample binary search tree

Suppose we remove the root node 8, and we wish to replace it with any node from the left subtree then what will be the new root?

Solution

The new node will be node 6. To maintain the properties of the binary search tree, the maximum value from the left subtree should be the new root.

Question 4

What will be the `inorder`, `postorder`, and `preorder` traversal of the following tree?

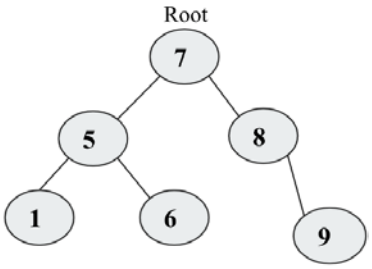

Figure A.6: Example tree

Solution

The `preorder` traversal will be `7-5-1-6-8-9`.

The `inorder` traversal will be `1-5-6-7-8-9`.

The `postorder` traversal will be `1-6-5-9-8-7`.

Question 5

How do you find out if two trees are identical?

Solution

In order to find out if two binary trees are identical or not, both of the trees should have exactly the same data and element arrangement. This can be done by traversing both of the trees with any of the traversal algorithms (it should be the same for both trees) and matching them element by element. If all the elements are the same in traversing both of the trees, then the trees are identical.

Question 6

How many leaves are there in the tree mentioned in *question 4*?

Solution

Three, nodes `1`, `6`, and `9`.

Question 7

What is the relation between a perfect binary tree's height and the number of nodes in that tree?

Solution

$\log_2 (n+1) = h$.

The number of nodes in each level:

Level `0`: $2^0 = 1$ nodes

Level `1`: $2^1 = 2$ nodes

Level `2`: $2^2 = 4$ nodes

Level `3`: $2^3 = 8$ nodes

The total nodes at level `h` can be computed by adding all nodes in each level:

$n = 2^0 + 2^1 + 2^2 + 2^3 + \ldots\ldots 2^{h-1} = 2^h - 1$

So, the relationship between `n` and `h` is: $n = 2^h - 1$

$= \log (n+1) = \log 2^h$

$= \log_2 (n+1) = h$

Chapter 7: Heaps and Priority Queues

Question 1

What will be the time complexity for deleting an arbitrary element from the `min-heap`?

Solution

To delete any element from the `heap`, we first have to search the element that is to be deleted, and then we delete the element.

Total time complexity = Time for searching the element + Deleting the element

= `O(n)` + `O(log n)`

= `O(n)`

Question 2

What will be the time complexity for finding the k^{th} smallest element from the `min-heap`?

Solution

The k^{th} element can be found out from the `min-heap` by performing `delete` operations k times. For each `delete` operation, the time complexity is `O(logn)`. So, the total time complexity for finding out the k^{th} smallest element will be `O(klogn)`.

Question 3

What will be the time complexity to make a `max-heap` that combines two `max-heap` each of size `n`?

Solution

`O(n)`.

Since the time complexity of creating a `heap` from `n` elements is `O(n)`, creating a `heap` of `2n` elements will also be `O(n)`.

Question 4

What will be the worst-case time complexity for ascertaining the smallest element from a binary max-heap and binary min-heap?

Solution

In a max-heap, the smallest element will always be present at a leaf node. So, in order to find out the smallest element, we have to search all the leaf nodes. So, the worst-case complexity will be `O(n)`.

The worst-case time complexity to find out the smallest element in the min-heap will be `O(1)` since it will always be present at the root node.

Question 5

The level order traversal of `max-heap` is `12`, `9`, `7`, `4`, `2`. After inserting new elements `1` and `8`, what will be the final `max-heap` and level order traversal of the final `max-heap`?

Solution

The `max-heap` after the insertion of element `1` is shown in the below figure:

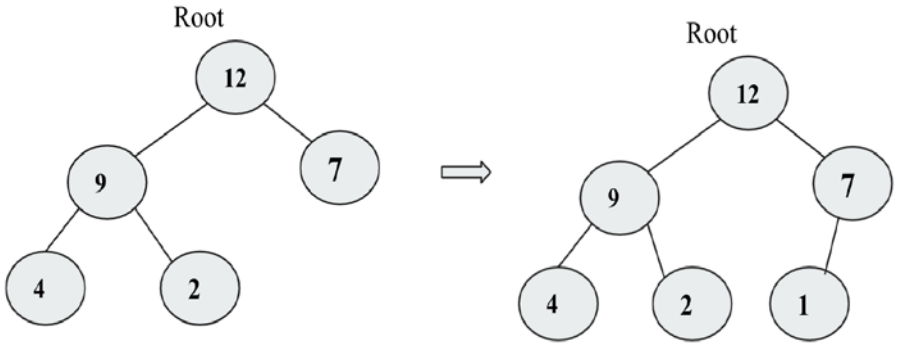

Figure A.7: The max-heap before insertion of element 8

The final `max-heap` after the insertion of element `8` is shown in the below figure:

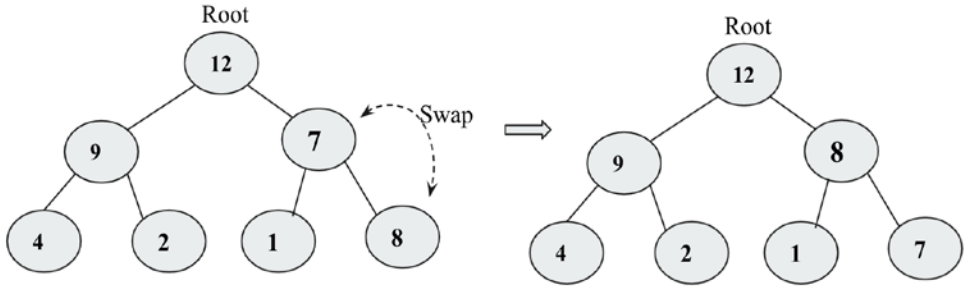

Figure A.8: The max-heap after the insertion of elements 1 and 8

The level order traversal of the final `max-heap` will be `12, 9, 8, 4, 2, 1, 7`.

Question 6

Which of the following is a binary `max-heap`?

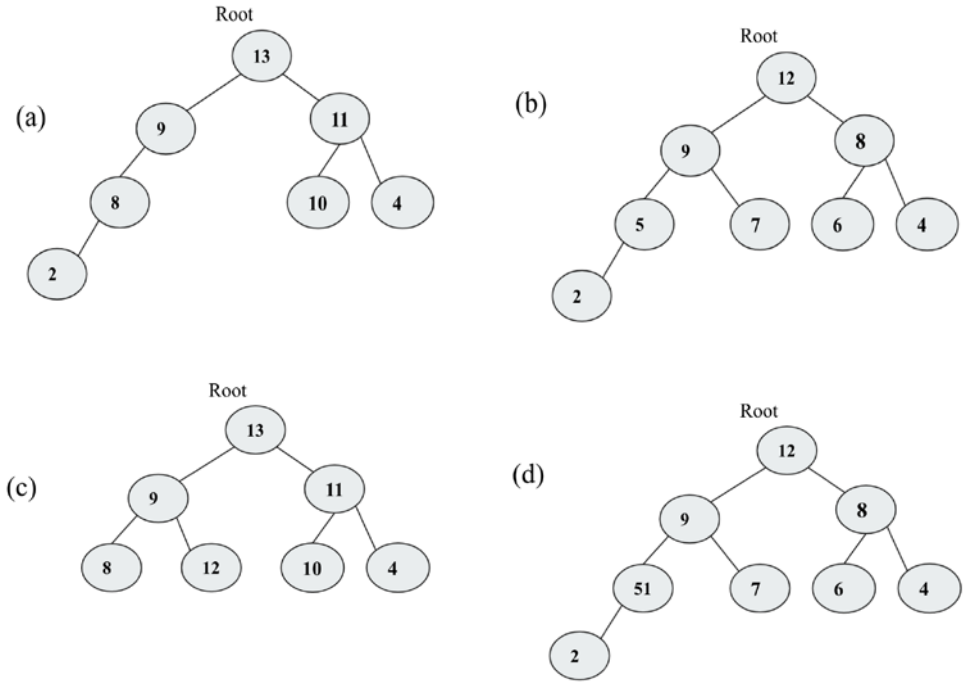

Figure A.9: Example trees

Solution

B.

A binary `max-heap` should be a complete binary tree and all the levels should be filled, except the last level. The value of the parent should be greater or equal to the values of its children.

Option A is not correct because it is not a complete binary tree. Options C and D are not correct because they are not fulfilling the `heap` property. Option B is correct because it is complete and fulfills the `heap` property.

Chapter 8: Hash Tables

Question 1

There is a hash table with 40 slots and there are 200 elements stored in the table. What will be the load factor of the hash table?

Solution

The load factor of the hash table = (no. of elements) / (no. of table slots) = 200/40 = 5.

Question 2

What is the worst-case search time of hashing using a separate chaining algorithm?

Solution

The worst-case time complexity for searching in a separate chaining algorithm using linked lists is `O(n)`, because in the worst case, all the items will be added to `index 1` in a linked list, searching an item will work similarly to a linked list.

Question 3

Assume a uniform distribution of keys in the hash table. What will be the time complexities for the `search`/`insert`/`delete` operations?

Solution

The index of the hash table is computed from the key in `O(1)` time when the keys are uniformly distributed in the hash table. The creation of the table will take `O(n)` time, and other operations such as `search`, `insert`, and `delete` operations will take `O(1)` time because all the elements are uniformly distributed, hence, we directly get the required element.

Question 4

What will be the worst-case complexity for removing the duplicate characters from an array of characters?

Solution

The brute force algorithm starts with the first character and searches linearly with all the characters of the array. If a duplicate character is found, then that character should be swapped with the last character and then the length of the string should be decremented by one. The same process is repeated until all characters are processed. The time complexity of this process is $O(n^2)$.

It can be implemented more efficiently using a hash table in `O(n)` time.

Using this method, we start with the first character of the array and store it in the hash table according to the hash value. We do it for all the characters. If there is any collision, then that character can be ignored, otherwise, the character is stored in the hash table.

Chapter 9: Graphs and Algorithms

Question 1

What is the maximum number of edges (without self-loops) possible in an undirected simple graph with five nodes?

Solution

Each node can be connected to every other node in the graph. So, the first node can be connected to `n-1` nodes, the second node can be connected to `n-2` nodes, the third node can be connected to `n-3` nodes, and so on. The total number of nodes will be:

[(n-1)+(n-2)+ ... +3+2+1] = n(n-1)/2.

Question 2

What do we call a graph in which all the nodes have an equal degree?

Solution

A complete graph.

Question 3

Explain what cut vertices are and identify the cut vertices in the given graph?

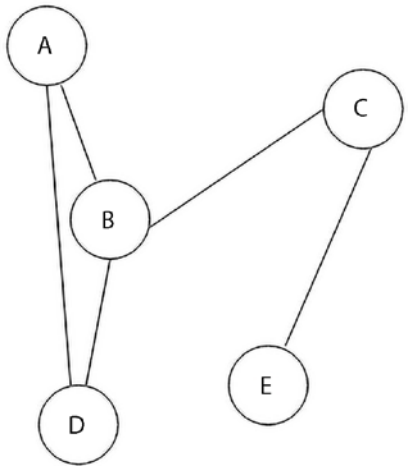

Figure A.10: Sample graph

Solution

Cut vertices also known as articulation points. These are the vertices in the graph, after removal of which, the graph splits in two disconnected components. In the given graph, the vertices `B`, and `C` are cut vertices since after removal of node `B`, the graph will split into `{A, D}`, `{C,E}` vertices. And, after removal of node `C`, the graph will split into `{A,B, D}`, `{E}` vertices.

Question 4

Assuming a graph `G` of order `n`, what will be the maximum number of cut vertices possible in graph `G`?

Solution

It will be `n-2`, since the first and last vertices will not be cut vertices, except those two nodes, all nodes can split the graph into two disconnected graphs. See the below graph:

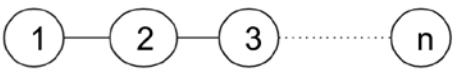

Figure A.11: A graph G

Chapter 10: Searching

Question 1

On average, how many comparisons are required in a linear search of n elements?

Solution

The average number of comparisons in linear search will be as follows. When a search element is found at the 1st position, 2nd position, 3rd position, and similarly at the nth position, correspondingly, it will require 1, 2, 3... n number of comparisons.

Total average number of comparisons

$$= \frac{(1 + 2 + 3 + \cdots n)}{n}$$

$$= \frac{[\frac{n(n+1)}{2}]}{n}$$

$$= \frac{(n+1)}{2}$$

Question 2

Assume there are eight elements in a sorted array. What is the average number of comparisons that will be required if all the searches are successful and if the binary search algorithm is used?

Solution

Average number of comparisons = (1+2+2+3+3+3+3+4)/8

= 21/8

= 2.625

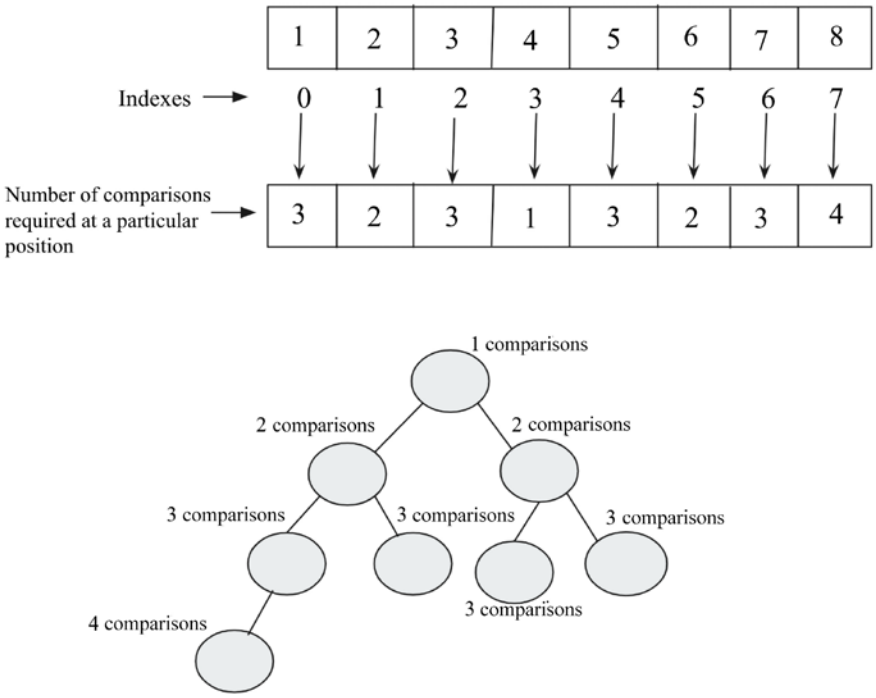

Figure A.12: Demonstration of number of the comparisons in the given array

Question 3

What is the worst-case time complexity of the binary search algorithm?

Solution

`O(logn)`.

The worst-case scenario of the binary search algorithm will occur when the desired element is present in the first position or at the last position. In that case, `log(n)` comparisons will be required. Hence the worst-case complexity will be `O(logn)`.

Question 4

When should the interpolation search algorithm perform better than the binary search algorithm?

Solution

The interpolation search algorithm performs better than the binary search algorithm when the data items in the array are uniformly distributed.

Chapter 11: Sorting

Question 1

If an array `arr = {55, 42, 4, 31}` is given and bubble sort is used to sort the array elements, then how many passes will be required to sort the array?

a. 3

b. 2

c. 1

d. 0

Solution

The answer is a. To sort `n` elements, the bubble sort algorithm requires `(n-1)` iterations (passes), where `n` is the number of elements in the given array. Here in the question, the value of `n = 4`, so `4-1 = 3` iterations will be required to sort the given array.

Question 2

What is the worst-case complexity of bubble sort?

a. O(nlogn)

b. O(logn)

c. O(n)

d. $O(n^2)$

Solution

The answer is d. The worst case appears when the given array is in reverse order. In that case, the time complexity of bubble sort would be $O(n^2)$.

Question 3

Apply quicksort to the sequence (56, 89, 23, 99, 45, 12, 66, 78, 34). What is the sequence after the first phase, and what pivot is the first element?

a. 45, 23, 12, 34, 56, 99, 66, 78, 89

b. 34, 12, 23, 45, 56, 99, 66, 78, 89

c. 12, 45, 23, 34, 56, 89, 78, 66, 99

d. 34, 12, 23, 45, 99, 66, 89, 78, 56

Solution

b.

After the first phase, 56 would be in the right position so that all the elements smaller than 56 will be on the left side of it, and elements bigger than 56 will be on the right side of it. Further, quicksort is applied recursively to the left subarray and right subarray. The process of the quicksort for the given sequence, as shown in the below figure.

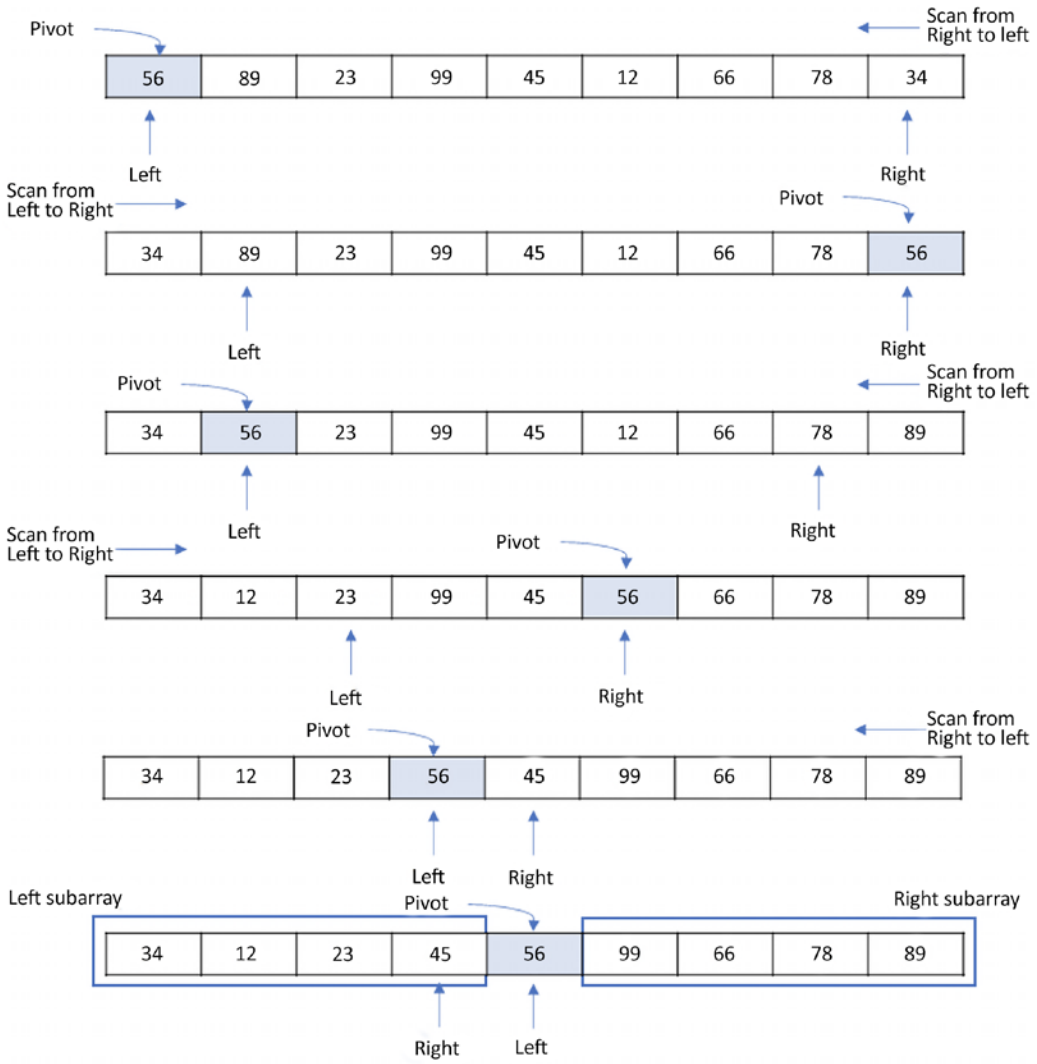

Figure A.13: Demonstration of the quicksort algorithm

Question 4

Quicksort is a _____

a. Greedy algorithm
b. Divide-and-conquer algorithm
c. Dynamic programming algorithm
d. Backtracking algorithm

Solution

The answer is b. Quicksort is a divide-and-conquer algorithm. Quick sort first partitions a large array into two smaller sub arrays and then recursively sorts the sub-arrays. Here, we find the pivot element such that all elements to the left side of the pivot element would be smaller than the pivot element and create the first subarray. The elements to the right side of the pivot element are greater than the pivot element and create the second subarray. Thus, the given problem is reduced into two smaller sets. Now, sort these two subarrays again, finding the pivot element in each subarray, i.e. apply quicksort on each subarray.

Question 5

Consider a situation where a `swap` operation is very costly. Which of the following sorting algorithms should be used so that the number of `swap` operations is minimized?

a. Heap sort
b. Selection sort
c. Insertion sort
d. Merge sort

Solution

b. In the selection sort algorithm, in general, we identify the largest element, and then swap it with the last element so that in each iteration, only one `swap` is required. For `n` elements, the total (`n-1`) swaps will be required, which is the lowest in comparison to all other algorithms.

Question 6

If the input array `A = {15, 9, 33, 35, 100, 95, 13, 11, 2, 13}` is given, using selection sort, what would be the order of the array after the fifth `swap`? (Note: it counts regardless of whether they exchange or remain in the same position.)

a. 2, 9, 11, 13, 13, 95, 35, 33, 15, 100

b. 2, 9, 11, 13, 13, 15, 35, 33, 95, 100

c. 35, 100, 95, 2, 9, 11, 13, 33, 15, 13

d. 11, 13, 9, 2, 100, 95, 35, 33, 13, 13

Solution

The answer is a. In selection sort, select the smallest element. Start the comparison from the beginning of the array and swap the smallest element with the first greatest element. Now, exclude the previous element that was chosen as the smallest element, as it has been put in the right place.

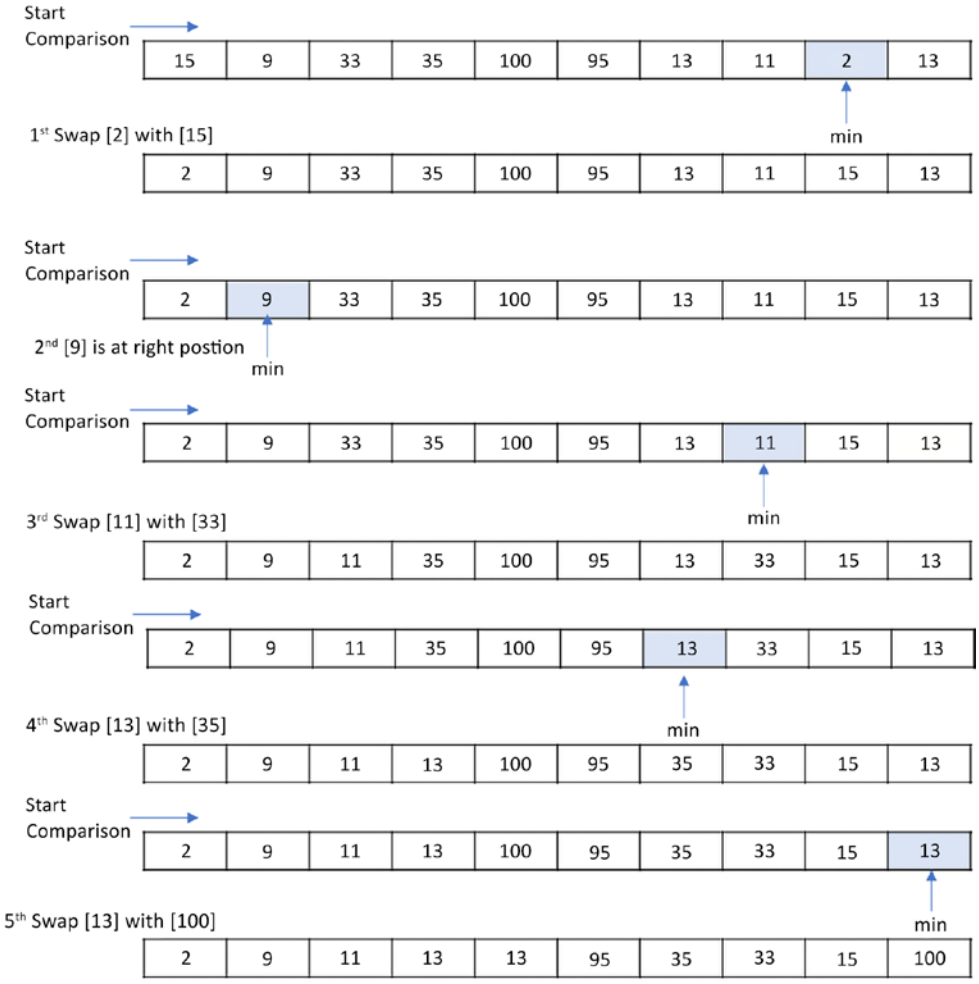

Figure A.14: Demonstration of insertion sort on the given sequence

Question 7

What will be the number of iterations to sort the elements `{44, 21, 61, 6, 13, 1}` using insertion sort?

a. 6

b. 5

c. 7

d. 1

Solution

The answer is a. Suppose there are `N` keys in an input list, then it requires `N` iterations to sort the entire list using insertion sort.

Question 8

How will the array elements `A= [35, 7, 64, 52, 32, 22]` look after the second iteration, if the elements are sorted using insertion sort?

a. 7, 22, 32, 35, 52, 64

b. 7, 32, 35, 52, 64, 22

c. 7, 35, 52, 64, 32, 22

d. 7, 35, 64, 52, 32, 22

Solutions

d. Here `N = 6`. In the first iteration, the first element, that is, `A[1] = 35`, is inserted into array `B`, which is initially empty. In the second iteration, `A[2] = 7` is compared with the elements in `B` starting from the rightmost element of `B` to find its place. So, after the second iteration, the input array would be `A = [7, 35, 64, 52, 32, 22]`.

Chapter 12: Selection Algorithm

Question 1

What will be the output if the quickselect algorithm is applied to the given array `arr=[3, 1, 10, 4, 6, 5]` with `k` given as 2?

Solution

1. Given the initial array: `[3, 1, 10, 4, 6, 5]`, we can find the median of medians: `4` (index = 3).
2. We swap the pivot element with the first element: `[4, 1, 3, 10, 6, 5]`.
3. We will move the pivot element to its correct position: `[1, 3, 4, 10, 6, 5]`.
4. Now we get a split index equal to `2` but the value of `k` is also equal to `2`, hence the value at index `2` will be our output. Hence the output will be `4`.

Question 2

Can quickselect find the smallest element in an array with duplicate values?

Solution

Yes, it works. By the end of every iteration, we have all elements less than the current pivot stored to the left of the pivot. Let's consider when all elements are the same. In this case, every iteration ends up putting a pivot element to the left of the array. And the next iteration will continue with one element shorter in the array.

Question 3

What is the difference between the quicksort algorithm and the quickselect algorithm?

Solution

In quickselect, we do not sort the array, and it is specifically for finding the k^{th} smallest element in the array. The algorithm repeatedly divides the array into two sections based on the value of the pivot element. As we know, the pivot element will be placed such that all the elements to its left are smaller than the pivot element, and all the elements to the right are larger than the pivot element. Thus, we can select any one of the segments of the array based on the target value. This way, the size of the operable range of our array keeps on reducing. This reduces the complexity from $O(n\log_2(n))$ to $O(n)$.

Question 4

What is the main difference between the deterministic selection algorithm and the quickselect algorithm?

Solution

In the `quickselect` algorithm, we find the k^{th} smallest element in an unordered list based on picking up the pivot element randomly. Whereas, in the deterministic selection algorithm, which is also used for finding the k^{th} smallest element from an unordered list, but in this algorithm, we choose a pivot element by using median of medians, instead of taking any random pivot element.

Question 5

What triggers the worst-case behavior of the selection algorithm?

Solution

Continuously picking the largest or smallest element on each iteration triggers the worst-case behavior of the selection algorithm.

Chapter 13: String Matching Algorithms

Question 1

Show the KMP `prefix` function for the pattern `"aabaabcab"`.

Solution

The `prefix` function values are given below:

pattern	a	a	b	a	a	b	c	a	b
`prefix_function` π	0	1	0	1	2	3	0	1	0

Table A.2: Prefix function for the given patten

Question 2

If the expected number of valid shifts is small and the modulus is larger than the length of the pattern, then what is the matching time of the Rabin-Karp algorithm?

a. Theta (`m`)
b. Big O (`n+m`)
c. Theta (`n-m`)
d. Big O (`n`)

Solution

Big O (`n+m`)

Question 3

How many spurious hits does the Rabin-Karp string matching algorithm encounter in the text `T = "3141512653849792"` when looking for all occurrences of the pattern `P = "26"`, working modulo `q = 11` and over the alphabet set `Σ = {0, 1, 2,..., 9}`?

Solution

2.

Question 4

What is the basic formula applied in the Rabin-Karp algorithm to get the computation time as Theta (m)?

a. Halving rule
b. Horner's rule
c. Summation lemma
d. Cancellation lemma

Solution

Horner's rule.

Question 5

The Rabin-Karp algorithm can be used for discovering plagiarism in text documents.

a. True
b. False

Solution

True, the Rabin-Karp algorithm is a string matching algorithm, and it can be used for detecting plagiarism in text documents.

Join our community on Discord

Join our community's Discord space for discussions with the author and other readers:

`https://packt.link/MEvK4`