ADA Question Bank

Ques 1: What is an algorithm.

Ans: An algorithm is a procedure used to solve a well defined computational problem.

Ques 2: Give the complexity in terms of Big O notation for the following algorithmic equations ?

(a) $5.n^2 – 6n + 2$ (b) $6.2^n + 6$

Ans. (a) $O(n^2)$

(b) $O(2^n)$

Ques 3. Arrange the following in increasing order of time complexity ?

$O(1)$, $O(2^n)$, $O(n^2)$, $O(n\log n)$, $O(n^3)$, $O(n)$, $O(\log n)$

Ans. $O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$

Ques 4. What do you understand by term O in Big O notation ?

Ans. $O(g(n)) = \{f(n)$: there exists positive constants c , n0 such that $0 <= f(n) <= cg(n)$ for all $n\}$

Ques 5. Show that the solution of $T(n)=T(n/2) + 1$ is $O(\lg n)$.

Ans. Using the iteration method, we substitute the function recursively till we arrive at $T(1)$.
i.e $T(n)=T(n/2) + 1 = T(n/4) + 1 + 1 = ...$

We get a total of $\lg(n)$ iterations
i.e $T(n)=T(1) +\lg(n)$

Thus the complexity is $O(\lg n)$.

Ques6. Show that the solution to T(n)=2*T((n/2)+17) is O(n*(lg n)).

Ans. Using the iteration method, it takes lg(n) iterations to reach T(1).After final iteration:

T(n)=(2^lg(n))*(T(1) + lg(n)*17)

Thus the complexity is (2^lg(n))*(lg n) = O(n*lg(n))

Ques7. Solve the recurrence T(n)=2*T(sqrt(n)) + 1.

Ans. Let n=2^m,then T(2^m)=2*T(2^m/2) + 1
Now let S(m)=T(2^m).
=>S(m)=2*S(m/2) +1

This is the same as 2nd problem.
Thus the complexity is O(m*lg(m)),but m=lg(n)

Thus the final complexity is O(lg(n)*lg(lg(n)))

Ques 8: Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?

Ans: No. Minimal spanning tree assures that the total weight of the tree is kept at its minimum. But it doesn't mean that the distance between any two nodes involved in the minimum-spanning tree is minimum.

Ques 9: What is the difference between BFS and DFS?

Ans : BFS: This can be throught of as being like Dijkstra's algorithm for shortest paths, but with every edge having the same length. However it is a lot simpler and doesn't need any data structures. We just keep a tree (the breadth first search tree), a list of nodes to be added to the tree, and markings (Boolean variables) on the vertices to tell whether they are in the tree or list.
Depth first search is another way of traversing graphs, which is closely related to preorder traversal of a tree. Recall that preorder traversal simply visits each node before its children. It is most easy to program as a recursive routine:

Ques 10:Write an algorithm to find the shortest path in Graphs ?

Ans: The Floyd–Warshall algorithm compares all possible paths through the graph between each pair of vertices.

Assume *edgeCost*(i,j) returns the cost of the edge from i to j (infinity if there is none), n is the number of vertices, and *edgeCost*(*i*,*i*) = 0.

```
Algorithm
int path[][]; // a 2-D matrix.
// At each step, path[i][j] is the (cost of the) shortest path
//  from i to j using intermediate vertices (1..k-1).
// Each path[i][j] is initialized to edgeCost (i,j)
//  or ∞ if there is no edge between i and j.

procedure FloydWarshall ()
  for k in 1..n
    for each pair (i,j) in {1,..,n}x{1,..,n}
        path[i][j] = min ( path[i][j], path[i][k]+path[k][j] );
```

Ques 11: Write a single source shortest path algorithm and its time complexity?

Ans : Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights. It is a greedy algorithm and similar to Prim's algorithm. Algorithm starts at the source vertex, s, it grows a tree, T, that ultimately spans all vertices reachable from S. Vertices are added to T in order of distance i.e., first S, then the vertex closest to S, then the next closest, and so on. Following implementation assumes that graph G is represented by adjacency lists.

```
DIJKSTRA (G, w, s)
INITIALIZE SINGLE-SOURCE (G, s)
S ← { }    // S will ultimately contains vertices of final shortest-path weights from s
Initialize priority queue Q i.e., Q ← V[G]
while priority queue Q is not empty do
   u ← EXTRACT_MIN(Q)   // Pull out new vertex
   S ← S È {u}
   // Perform relaxation for each vertex v adjacent to u
   for each vertex v in Adj[u] do
      Relax (u, v, w)
```

Analysis
Like Prim's algorithm, Dijkstra's algorithm runs in O(|E|lg|V|) time.

Ques 12:Does dijkstra's algorithm for shortest path provide the optimal solution ?

Ans : Yes, dijkstra's algorithm for shortest path provide the optimal solution.
To find the shortest path between points, the weight or length of a path is calculated as the sum of the weights of the edges in the path. A path is a shortest path is there is no path from x to y with lower weight. Dijkstra's algorithm finds the shortest path from x to y in order of increasing distance from x. That is, it chooses the first minimum edge, stores

this value and adds the next minimum value from the next edge it selects. It starts out at one vertex and branches out by selecting certain edges that lead to new vertices. It is similar to the minimum spanning tree algorithm, in that it is "greedy", always choosing the closest edge in hopes of an optimal solution.

Ques 13:Explain Prim’s Algorithm .

Ans: Like Kruskal's algorithm, Prim's algorithm is based on a generic MST algorithm. The main idea of Prim's algorithm is similar to that of Dijkstra's algorithm for finding shortest path in a given graph. Prim's algorithm has the property that the edges in the set A always form a single tree. We begin with some vertex *v* in a given graph G =(V, E), defining the initial set of vertices A. Then, in each iteration, we choose a minimum-weight edge (*u*, *v*), connecting a vertex v in the set A to the vertex *u* outside of set A. Then vertex *u* is brought in to A. This process is repeated until a spanning tree is formed. Like Kruskal's algorithm, here too, the important fact about MSTs is we always choose the smallest-weight edge joining a vertex inside set A to the one outside the set A. The implication of this fact is that it adds only edges that are safe for A; therefore when the algorithm terminates, the edges in set A form a MST.

Algorithm

```
MST_PRIM (G, $w$, $v$)
 1.  Q ← V[G]
 2.  for each $u$ in Q do
 3.      key [$u$] ← ∞
 4.  key [$r$] ← 0
 5.  π[$r$] ← NIl
 6.  while queue is not empty do
 7.      $u$ ← EXTRACT_MIN (Q)
 8.      for each $v$ in Adj[$u$] do
 9.          if v is in Q and $w(u, v) < key [v]$
 10.             then π[$v$] ← $w(u, v)$
 11.                 key [$v$] ← $w(u, v)$
```

Ques 14 :What is the difference between binary searching and Fibonacci search?

Ans: Binary Search: Binary search is the process of locating an element in a sorted list. The search starts by dividing the list into two parts. The algorithm compares the median value. If the search element is less than the median value, the top list only will be searched, after finding the middle element of that list. The process continues until the element is found or the search in the top list is completed. The same process is continued for the bottom list, until the element is found or the search in the bottom list is completed. If an element is found that must be the median value.

Fibonacci Search: Fibonacci search is a process of searching a sorted array by utilizing divide and conquer algorithm. Fibonacci search examines locations whose addresses have lower dispersion. When the search element has non-uniform access memory storage, the Fibonacci search algorithm reduces the average time needed for accessing a storage location.

Ques 15: Explain the bubble sort algorithm.

Ans: Bubble sort algorithm is used for sorting a list. It makes use of a temporary variable for swapping. It compares two numbers at a time and swaps them if they are in wrong order. This process is repeated until no swapping is needed. The algorithm is very inefficient if the list is long.

E.g. List: - 7 4 5 3

1. 7 and 4 are compared

2. Since 4 < 7, 4 is stored in a temporary variable.

3. the content of 7 is now stored in the variable which was holding 4

4. Now, the content of temporary variable and the variable previously holding 7 are swapped.

Ques 16: Explain Quick Sort

Ans: The quick sort algorithm is of the divide and conquer type. That means it works by reducing a sorting problem into several easier sorting problems and solving each of them. A dividing value is chosen from the input data, and the data is partitioned into three sets: elements that belong before the dividing value, the value itself, and elements that come after the dividing value. The partitioning is performed by exchanging elements that are in the first set but belong in the third with elements that are in the third set but belong in the first Elements that are equal to the dividing element can be put in any of the three sets the algorithm will still work properly.

Ques 17: Explain Merge Sort

Ans: The merge sort is a divide and conquer sort as well. It works by considering the data to be sorted as a sequence of already-sorted lists (in the worst case, each list is one element long). Adjacent sorted lists are merged into larger sorted lists until there is a single sorted list containing all the elements. The merge sort is good at sorting lists and other data structures that are not in arrays, and it can be used to sort things that don't fit into memory. It also can be implemented as a stable sort.

Ques 18:Sort the given values using Quick Sort?

65	70	75	80	85	60	55	50	45

Ans:

Sorting takes place from the pivot value, which is the first value of the given elements, this is marked bold. The values at the left pointer and right pointer are indicated using L and R respectively.

65	70^L	75	80	85	60	55	50	45^R

Since pivot is not yet changed the same process is continued after interchanging the values at L and R positions

65	45	75^L	80	85	60	55	50^R	70

65	45	50	80^L	85	60	55^R	75	70

65	45	50	55	85^L	60^R	80	75	70

65	45	50	55	60^R	85^L	80	75	70

When the L and R pointers cross each other the pivot value is interchanged with the value at right pointer. If the pivot is changed it means that the pivot has occupied its original position in the sorted order (shown in bold italics) and hence two different arrays are formed, one from start of the original array to the pivot position-1 and the other from pivot position+1 to end.

60^L	45	50	55^R	65	85^L	80	75	70^R

55^L5	45	50^R	60	65	70^R	80^L	75	85

50^L	45^R	55	60	65	70	80^L	75^R	85

In the next pass we get the sorted form of the array.

45	50	55	60	65	70	75	80	85

Ques 19: What is difference between Bubble Sort and Quick sort?

Ans: Bubble Sort: The simplest sorting algorithm. It involves the sorting the list in a repetitive fashion. It compares two adjacent elements in the list, and swaps them if they are not in the designated order. It continues until there are no swaps needed. This is the signal for the list that is sorted. It is also called as comparison sort as it uses comparisons.

Quick Sort: The best sorting algorithm which implements the 'divide and conquer' concept. It first divides the list into two parts by picking an element a 'pivot'. It then arranges the elements those are smaller than pivot into one sub list and the elements those are greater than pivot into one sub list by keeping the pivot in its original place.

Ques 20. What are elements of dynamic programming?

Ans. Elements of Dynamic Programming
• Optimal sub-structure (Principle of Optimality)
an optimal solution to the problem contains within it optimal solutions to sub-problems.
• Overlapping subproblems
there exist some places where we solve the same subproblem more than once.

Ques 21. What is a greedy algorithm?

Ans. A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum. On some problems, a greedy strategy need not produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution.

Ques22. What is difference between dynamic programming and greedy algorithm?

|Ans. Dynamic Programming
Bottom up (while Greedy is top-down)
• Dynamic programming can be overkill; greedy algorithms tend to be easier to code

|Ques 23. What is Matrix Chain Multiplication Problem?

Ans. Suppose we have a sequence or chain $A_1, A_2, \ldots, A_n$ of *n* matrices to be multiplied.
There are many possible ways (parenthesizations) to compute the product
Example: consider the chain $A_1, A_2, A_3, A_4$of 4 matrices
Let us compute the product $A_1A_2A_3A_4$
There are 5 possible ways:
$(A_1 (A_2 (A_3 A_4)))$

$(A_1 ((A_2 A_3) A_4))$
$((A_1 A_2)(A_3 A_4))$
$((A_1 (A_2 A_3)) A_4)$
$(((A_1 A_2) A_3) A_4)$
Matrix-chain multiplication problem states that given a chain $A_1, A_2, \ldots, A_n$ of *n* matrices, where for *i*=1, 2, …, *n*, matrix A*i* has dimension $p_{i-1} \times p_i$. Parenthesize the product $A_1, A_2, \ldots, A_n$ such that the total number of scalar multiplications is minimized.

Ques 24. Given the following 2 strings S and T, find their longest common subsequence.
S = ABAZDC
T = BACBAD

Ans.

	B	A	C	B	A	D
A	0	1	1	1	1	1
B	1	1	1	2	2	2
A	1	2	2	2	3	3
Z	1	2	2	2	3	3
D	1	2	2	2	3	4
C	1	2	3	3	3	4

Thus LCS is 4.

Ques 25. What is the time complexity for matrix multiplication and Strassen's Matrix Multiplication?

Ans. Matrix Multiplication=Θ (n^3)
Strassen's Matrix Multiplication: Θ ($n^{2.81}$)

Ques 26. What is activity selection problem?

Ans. Scheduling a resource among several competing activities
- Definition
  - S={1, 2,…, n} – activities that wish to use a resource and a finish time fi, where si ≤ fi
  - If selected, activity i takes place during the half-open interval [si, fi)
  - Activities i and j are compatible if [si, fi) and [sj, fj) don't overlap
  - The activity-selection problem is to select a maximum-size set of mutually compatible activities

Ques27. Explain Strassen's Matrix Multiplication.

Ans. It uses only 7 scalar multiplications to multiply two 2x2 matrices. It divides each of A,B and C matrices into four n/2 x n/2 matrices and rewriting equation C=AB as:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

r= P5+P4-P2+P6

s=P1+P2

t=P3+P4

u=P5+P1-P3-P7

Where:

P1=a(f-h)

P2=(a+b)h

P3=(c+d)e

P4=d(g-e)

P5=(a+d)(e+h)

P6=(b-d)(g+h)

P7=(a-c)(e+f)

Ques 28. Write the algorithm for Topological Sort.

```
Ans. int topologicalOrderTraversal( ){
    int numVisitedVertices = 0;
    while(there are more vertices to be visited){
        if(there is no vertex with in-degree 0)
            break;
        else{
  select a vertex v that has in-degree 0;
  visit v;
  numVisitedVertices++;
  delete v and all its emanating edges;
        }
    }

   return numVisitedVertices;
}
```

Ques 29.Compare the running time for Naïve String Matching and Knuth Morris Pratt algorithm for string Matching.

Ans. Running time for Naïve String Matching : O((n-m+1) m)
Running time for KMP Matcher: O((n-m+1) m)

Ques 30. What is optimal binary search tree?

Ans. OBST is one special kind of advanced tree. It focus on how to reduce the cost of the search of the BST. It needs 3 tables to record probabilities, cost, and root.
It has n keys (representation k1,k2,…,kn) in sorted order (so that k1<k2<…<kn), and we wish to build a binary search tree from these keys. For each ki ,we have a probability pi that a search will be for ki.
In contrast of, some searches may be for values not in ki, and so we also have n+1 "dummy keys" d0,d1,…,dn representating not in ki.
In particular, d0 represents all values less than k1, and dn represents all values greater than kn, and for i=1,2,…,n-1, the dummy key di represents all values between ki and ki+1.

Ques 31. What is the difference between divide and conquer and dynamic programming?

Ans. Both solve problem by dividing prob into sub-problems
(1)but in Dynamic programming the sub problems are not independent where as in divide and conquer the sub problems are independent of each other
(2)Dynamic programming solves the sub problem only once and then stores it in table whereas divide and conquer does more work on the sub problems and hence has more time consumption.

Ques 32. Consider an array with numbers 5 1 4 2 8 and sort using bubble sort.

Ans. First Pass:
(5 1 4 2 8) → (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps them.
(1 5 4 2 8) → (1 4 5 2 8), Swap since 5 > 4
(1 4 5 2 8) → (1 4 2 5 8), Swap since 5 > 2
(1 4 2 5 8) → (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.
Second Pass:
(1 4 2 5 8) → (1 4 2 5 8)
(1 4 2 5 8) → (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) → (1 2 4 5 8)
(1 2 4 5 8) → (1 2 4 5 8)
Now, the array is already sorted, but our algorithm does not know if it is completed. The

algorithm needs one whole pass without any swap to know it is sorted.
Third Pass:
(1 2 4 5 8) $\rightarrow$ (1 2 4 5 8)
(1 2 4 5 8) $\rightarrow$ (1 2 4 5 8)
(1 2 4 5 8) $\rightarrow$ (1 2 4 5 8)
(1 2 4 5 8) $\rightarrow$ (1 2 4 5 8)

Ques 33. Following are 7 activities with their respective deadlines and penalities.

ai 1 2 3 4 5 6 7
di 4 2 4 3 1 4 6
wi 70 60 50 40 30 20 10
 Solve using the task scheduling such that penalty is minimized.

Ans. The solution to this problem is {a2,a4,a1,a3,a7,a5,a6}
penality , w5+w6=50

Ques 34. What is Best First Search?

Ans. Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.

Efficient selection of the current best candidate for extension is typically implemented using a priority queue.

Ques 35. What is the difference between DFS and BFS.

Ans. DFS (Depth First Search) and BFS (Breadth First Search) are search algorithms used for graphs and trees. When you have an ordered tree or graph, like a BST, it's quite easy to search the data structure to find the node that you want. But, when given an unordered tree or graph, the BFS and DFS search algorithms can come in handy to find what you're looking for. The decision to choose one over the other should be based on the type of data that one is dealing with.

In a breadth first search, you start at the root node, and then scan each node in the first level starting from the leftmost node, moving towards the right. Then you continue scanning the second level (starting from the left) and the third level, and so on until you've scanned all the nodes, or until you find the actual node that you were searching for. In a BFS, when traversing one level, we need some way of knowing which nodes to traverse once we get to the next level. The way this is done is by storing the pointers to a level's child nodes while searching that level. The pointers are stored in FIFO (First-In-First-Out) queue. This, in turn, means that BFS uses a large amount of memory because we have to store the pointers.

Ques 36. What is Knapsack problem?

Ans. The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the count of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most useful items.

In the following, we have *n* kinds of items, 1 through *n*. Each kind of item *i* has a value v_i and a weight w_i. We usually assume that all values and weights are nonnegative. To simplify the representation, we can also assume that the items are listed in increasing order of weight. The maximum weight that we can carry in the bag is *W*.

The most common formulation of the problem is the 0-1 knapsack problem, which restricts the number x_i of copies of each kind of item to zero or one. Mathematically the 0-1-knapsack problem can be formulated as:

- maximize $\sum^{n} v_i$
- subject to $\sum^{n} w_i x_i \leqslant W,$

The bounded knapsack problem restricts the number x_i of copies of each kind of item to a maximum integer value c_i. Mathematically the bounded knapsack problem can be formulated as:

- maximize $\sum^{n} v_i$
- subject to $\sum^{n} w_i x_i \leqslant W, \qquad x_i \in$

The unbounded knapsack problem (UKP) places no upper bound on the number of copies of each kind of item.

Of particular interest is the special case of the problem with these properties:

- it is a decision problem,
- it is a 0-1 problem,
- for each kind of item, the weight equals the value: $w_i = v_i$.

Ques 37. What is the dynamic programmic solution to Knapsack problem?

Ans. If all weights ($w_1, \ldots, w_n, W$) are nonnegative integers, the knapsack problem can be solved in pseudo-polynomial time using dynamic programming. The following describes a dynamic programming solution for the *unbounded* knapsack problem.

To simplify things, assume all weights are strictly positive ($w_i > 0$). We wish to maximize total value subject to the constraint that total weight is less than or equal to W. Then for each $w \leq W$, define *m[w]* to be the maximum value that can be attained with total weight less than or equal to w. *m[W]* then is the solution to the problem.

Observe that *m[w]* has the following properties:

- $m[0] = 0$(the sum of zero items, i.e., the summation of the empty set)
- $m[w] = \max_{w_i \leq w}(v_i + m[w - w_i])$

where v_i is the value of the i-th kind of item.

0-1 knapsack problem

A similar dynamic programming solution for the *0-1 knapsack problem* also runs in pseudo-polynomial time. As above, assume $w_1, w_2, \ldots, w_n, W$ are strictly positive integers. Define $m[i,w]$ to be the maximum value that can be attained with weight less than or equal to w using items up to i.

We can define $m[i,w]$ recursively as follows:

- $m[0, w] = 0$
- $m[i, 0] = 0$
- $m[i, w] = m[i-1, w]$ if $w_i > w$(the new item is more than the current weight limit)
- $m[i, w] = \max(m[i-1, w], m[i-1, w-w_i] + v_i)$ if $w_i \leqslant w$.

Ques 38. What is Tower of Hanoi problem?

Ans. The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

- Only one disk may be moved at a time.

- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

The puzzle can be played with any number of disks, although many toy versions have around seven to nine of them. The game seems impossible to many novices, yet is solvable with a simple algorithm. The number of moves required to solve a Tower of Hanoi puzzle is 2^n-1, where *n* is the number of disks.

Ques 39. Sort the following using Selection sort:

64 25 12 22 11

Ans. 64 25 12 22 11

11 25 12 22 64

11 12 25 22 64

11 12 22 25 64

11 12 22 25 64

Ques 40. What is shell sort?

Ans. Shell sort, also known as Shell sort or Shell's method is an in-place comparison sort. It generalizes an exchanging sort, such as insertion or bubble sort, by allowing the comparison and exchange of elements that lie far apart . Its first version was published by Donald Shell in 1959 .The running time of Shell sort is heavily dependent on the gap sequence it uses. For many practical variants, determining their time complexity remains an open problem.

Shell sort is a multi-pass algorithm. Each pass is an insertion sort of the sequences consisting of every *h*-th element for a fixed gap *h* (also known as the increment). This is referred to as *h*-sorting.

Ques 41. Write the algorithm for counting sort.

Ans. ''' allocate an array Count[0..k] ; initialize each array cell to zero ; THEN '''

```
for each input item x:
   Count[key(x)] = Count[key(x)] + 1
total = 0
for i = 0, 1, ... k:
   c = Count[i]
```

```
    Count[i] = total
    total = total + c

''' allocate an output array Output[0..n-1] ; THEN '''
for each input item x:
    store x in Output[Count[key(x)]]
    Count[key(x)] = Count[key(x)] + 1
return Output
```

Ques 42. What is the difference between quick sort and randomized quick sort?

Ans. In quick sort the first element in the array is chosen as the pivot but in randomized quick sort any number in the array is chosen as the pivot.

Ques 43. What is difference between Bubble sort and Exchange sort?

Ans. The difference between these two sorts is the manner in which they compare the elements. The exchange sort compares the first element with each following element of the array, making any necessary swaps.

Ques 44. Sort the following using Exchange sort in descending order.

84	69	76	86	94	91

Array at beginning:	84	69	76	86	94	91
After Pass #1:	94	69	76	84	86	91
After Pass #2:	94	91	69	76	84	86
After Pass #3:	94	91	86	69	76	84
After Pass #4:	94	91	86	84	69	76
After Pass #5	94	91	86	84	76	69

Ques 45. Discuss the best case and worst case running time of insertion sort.

Ans. The best case input is an array that is already sorted. In this case insertion sort has a linear running time (i.e., $\Theta(n)$). During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted subsection of the array.

The worst case input is an array sorted in reverse order. In this case every iteration of the inner loop will scan and shift the entire sorted subsection of the array before inserting the next element. For this case insertion sort has a quadratic running time (i.e., $O(n^2)$).