匠人学院
jiangren.com.au

算法面试宝典

匠人学院简介

▪ 口号：拿offer，找匠人

▪ 定位：最大的华人找工作平台

▪ 愿景：让所有的海外华人找工作不再是难事

▪ 匠人由 匠人学院（JR Academy），匠人猎头（JR Talent），匠人科技（JR Solutions），IT匠人圈（社群）组成。

▪ 目前是澳洲最大华人IT组织。也得到了来自全球顶尖的互联网公司的认可，老人带新人。

▪ 匠人学院（JR Academy)是一家致力于为澳洲华人学业及就业保驾护航的平台，专注于帮助学员进行职业规划、技能提升、拓展职业社交圈，让所以澳洲华人寻找到理想的岗位，让所有企业都能聘请到优秀的人才。

联系我们 | 0421-672-555 | career@jiangren.com.au hello@jiangren.com.au | jracademy

匠人学院简介

1. 技能培训

百位业界导师制定了多方向课程

面向澳洲在校生，毕业生，转专业者

提供多座城市面授/远程教学

详细基础讲解+商业项目实战，带你从零基础到就业

2. 大学辅导

涵盖全澳范围内大学课程

面向/IT/IS/CS/商科等专业在校生

200+学霸老师 面授/远程保你学业无忧

课程专人负责 深挖课程考点难点

定制化辅导 答疑解惑 冲击高分

3. 求职就业

从面试官，雇主角度解决就业问题

100+业界精英组成的庞大导师队伍

拥有澳洲最专业的华人求职服务

与多所澳洲知名高校就业服务部门直接合作

综合提高学员能力，助理Offer

联系我们 0421-672-555 career@jiangren.com.au hello@jiangren.com.au jracademy

面试题

1. What is an algorithm? What is the need for an algorithm?

An algorithm is a well-defined computational procedure that takes some values or the set of values, as an input and produces a set of values or some values, as an output.

need for algorithm:
The algorithm provides the basic idea of the problem and an approach to solve it. Some reasons to use an algorithm are as follows.

- The algorithm improves the efficiency of an existing technique.
- To compare the performance of the algorithm with respect to other techniques.
- The algorithm gives a strong description of requirements and goal of the problems to the designer.
- The algorithm provides a reasonable understanding of the flow of the program.
- The algorithm measures the performance of the methods in different cases (Best cases, worst cases, average cases).
- The algorithm identifies the resources (input/output, memory) cycles required by the algorithm.
- With the help of an algorithm, we can measure and analyze the complexity time and space of the problems.
- The algorithm also reduces the cost of design.

面试题

2. What is the Complexity of Algorithm?

The complexity of the algorithm is a way to classify how efficient an algorithm is compared to alternative ones. Its focus is on how execution time increases with the data set to be processed. The computational complexity of the algorithm is important in computing.
It is very suitable to classify algorithm based on the relative amount of time or relative amount of space they required and specify the growth of time/ space requirement as a function of input size

Time complexity
Time complexity is a Running time of a program as a function of the size of the input.

Space complexity
Space complexity analyzes the algorithms, based on how much space an algorithm needs to completeits task. Space complexity analysis was critical in the early days of computing (when storage space on the computer was limited). Nowadays, the problem of space rarely occurs because space on the computer is broadly enough. We achieve the following types of analysis for complexity.

Worst-case: f(n)
It is defined by the maximum number of steps taken on any instance of size n.
Best-case: f(n)
It is defined by the minimum number of steps taken on any instance of size n.
Average-case: f(n)
It is defined by the average number of steps taken on any instance of size n.

面试题

3. Write an algorithm to reverse a string. For example, if my string is "uhsnamiH" then my result will be "Himanshu".

Algorithm to reverse a string.

Step1: start

Step2: Take two variable i and j

Step3: do length (string)-1, to set J at last position

Step4: do string [0], to set i on the first character.

Step5: string [i] is interchanged with string[j]

Step6: Increment i by 1

Step7: Increment j by 1

Step8: if i>j then go to step3

Step9: Stop

面试题

4.Write an algorithm to insert a node in a sorted linked list.

Algorithm to insert a node in a sorted linked list.
Case1:
Check if the linked list is empty then set the node as head and return it.
1. New_node-> Next= head;
2. Head=New_node

Case2:
Insert the new node in middle
1. While(P!= insert position)
2. {
3. P= p-> Next;
4. }
5. Store_next=p->Next;
6. P->Next= New_node;
7. New_Node->Next = Store_next;

Case3:
Insert a node at the end
1. While (P->next!= null)
2. {
3. P= P->Next;
4. }
5. P->Next = New_Node;
6. New_Node->Next = null;

面试题

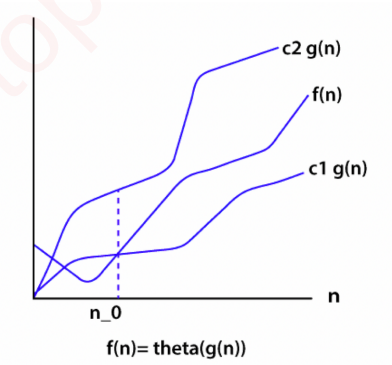
5. What are the Asymptotic Notations?

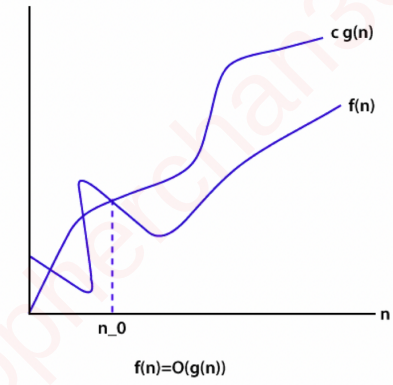
Asymptotic analysis is used to measure the efficiency of an algorithm that doesn't depend on machine-specific constants and prevents the algorithm from comparing the time taking algorithm. Asymptotic notation is a mathematical tool that is used to represent the time complexity of algorithms for asymptotic analysis.

The three most used asymptotic notation is as follows.

θ Notation

θ Notation defines the exact asymptotic behavior. To define a behavior, it bounds functions from above and below. A convenient way to get Theta notation of an expression is to drop low order terms and ignore leading constants.



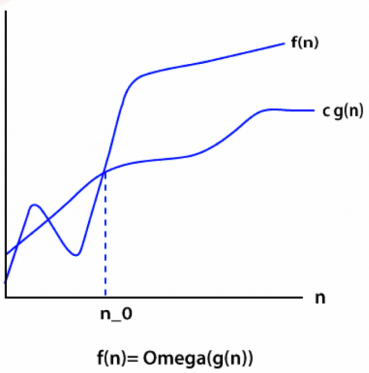


Big O Notation

The Big O notation bounds a function from above, it defines an upper bound of an algorithm. Let's consider the case of insertion sort; it takes linear time in the best case and quadratic time in the worst case. The time complexity of insertion sort is O(n2). It is useful when we only have upper bound on time complexity of an algorithm.

Ω Notation

Just like Big O notation provides an asymptotic upper bound, the Ω Notation provides an asymptotic lower bound on a function. It is useful when we have lower bound on time complexity of an algorithm.



6. Explain the Bubble sort algorithm?

Bubble sort is the simplest sorting algorithm among all sorting algorithm. It repeatedly works by swapping the adjacent elements if they are in the wrong order.
e.g.
(72538) we have this array for sorting.
Pass1:
(72538) -> (27538) swap 7 and 2.
(27538) -> (25738) swap 7 and 5
(25738) -> (25378) swap 7 and 3.
(25378) -> (25378) algorithm does not swap 7 and 8 because 7<8.
Pass2:
(25378) -> (25378) algorithm does not swap 2 and 5 because 2<5.
(25378) -> (23578) swap 3 and 5.
(23578) -> (23578) algorithm does not swap 5 and 7 because 5<7.
(23578) -> (23578) algorithm does not swap 7 and 8 because 7<8.
Here, the sorted element is (23578).

7. Explain the Bubble sort algorithm?

Dijkstra's algorithm is an algorithm for finding the shortest path from a starting node to the target node in a weighted graph. The algorithm makes a tree of shortest paths from the starting vertex and source vertex to all other nodes in the graph. Suppose you want to go from home to office in the shortest possible way. You know some roads are heavily congested and challenging to use this, means these edges have a large weight.
In Dijkstra's algorithm, the shortest path tree found by the algorithm will try to avoid edges with larger weights.

8. How to swap two integer without swapping the temporary variable in Java?

It's a very commonly asked trick question. There are many ways to solve this problem. But the necessary condition is we have to solve it without swapping the temporary variable. If we think about integer overflow and consider its solution, then it creates an excellent impression in the eye of interviewers.
Suppose we have two integers I and j, the value of i=7 and j=8 then how will you swap them without using a third variable. This is a journal problem. We need to do this using Java programming constructs. We can swap numbers by performing some mathematical operations like addition, subtraction, multiplication, and division. But maybe it will create the problem of integer overflow.

Using addition and subtraction

1. a= a + b;
2. b=a - b; // this will act like (a+b)-b, now b is equal to a.
3. a=a - b; // (a+b)-a, now, a is equal to b.

It is a nice trick. But in this trick, the integer will overflow if the addition is more than the maximum value of int primitive as defined by Integer.MAX_VALUE and if subtraction is less than minimum value i.e., Integer.MIN_VALUE.

Using XOR trick

Another solution to swap two integers without using a third variable (temp variable) is widely recognized as the best solution, as it will also work in a language which doesn't handle integer overflow like Java example C, C++. Java supports several bitwise operators. One of them is XOR (denoted by ^).

1. x=x^y;
2. y=x^y;
3. x=x^y;

面试题

9.What is a Hash Table? How can we use this structure to find all anagrams in a dictionary?

A Hash table is a data structure for storing values to keys of arbitrary type. The Hash table consists of an index into an array by using a Hash function. Indexes are used to store the elements. We assign each possible element to a bucket by using a hash function. Multiple keys can be assigned to the same bucket, so all the key and value pairs are stored in lists within their respective buckets. Right hashing function has a great impact on performance.

To find all anagrams in a dictionary, we have to group all words that contain the same set of letters in them. So, if we map words to strings representing their sorted letters, then we could group words into lists by using their sorted letters as a key.

```
1. FUNCTION find_anagrams(words)
2.     word_groups = HashTable<String, List>
3.     FOR word IN words
4.         word_groups.get_or_default(sort(word), []).push(word)
5.     END FOR
6.     anagrams = List
7.     FOR key, value IN word_groups
8.         anagrams.push(value)
9.     END FOR
10.    RETURN anagrams
```

The hash table contains lists mapped to strings. For each word, we add it to the list at the suitable key, or create a new list and add it to it.

10. Explain the BFS algorithm?

BFS (Breadth First Search) is a graph traversal algorithm. It starts traversing the graph from the root node and explores all the neighboring nodes. It selects the nearest node and visits all the unexplored nodes. The algorithm follows the same procedure for each of the closest nodes until it reaches the goal state.

Algorithm

Step1: Set status=1 (ready state)
Step2: Queue the starting node A and set its status=2, i.e. (waiting state)
Step3: Repeat steps 4 and 5 until the queue is empty.
Step4: Dequeue a node N and process it and set its status=3, i.e. (processed state)
Step5: Queue all the neighbors of N that are in the ready state (status=1) and set their status =2 (waiting state)
[Stop Loop]
Step6: Exit

11. What is Dijkstra's shortest path algorithm?

Dijkstra's algorithm is an algorithm for finding the shortest path from a starting node to the target node in a weighted graph. The algorithm makes a tree of shortest paths from the starting vertex and source vertex to all other nodes in the graph. Suppose you want to go from home to office in the shortest possible way. You know some roads are heavily congested and challenging to use this, means these edges have a large weight. In Dijkstra's algorithm, the shortest path tree found by the algorithm will try to avoid edges with larger weights.

12. Give some examples of Divide and Conquer algorithm?

Some problems that use Divide and conquer algorithm to find their solution are listed below.

- Merge Sort
- Quick Sort
- Binary Search
- Strassen's Matrix Multiplication
- Closest pair (points)

面试题

13. What are Greedy algorithms? Give some example of it?

A greedy algorithm is an algorithmic strategy which is made for the best optimal choice at each sub stage with the goal of this, eventually leading to a globally optimum solution. This means that the algorithm chooses the best solution at the moment without regard for consequences.
In other words, an algorithm that always takes the best immediate, or local, solution while finding an answer.
Greedy algorithms find the overall, ideal solution for some idealistic problems, but may discover less-than-ideal solutions for some instances of other problems.
Below is a list of algorithms that finds their solution with the use of the Greedy algorithm.
o Travelling Salesman Problem
o Prim's Minimal Spanning Tree Algorithm
o Kruskal's Minimal Spanning Tree Algorithm
o Dijkstra's Minimal Spanning Tree Algorithm
o Graph - Map Coloring
o Graph - Vertex Cover
o Knapsack Problem
o Job Scheduling Problem

14. What is a linear search?

Linear search is used on a group of items. It relies on the technique of traversing a list from start to end by visiting properties of all the elements that are found on the way. For example, suppose an array of with some integer elements. You should find and print the position of all the elements with their value. Here, the linear search acts in a flow like matching each element from the beginning of the list to the end of the list with the integer, and if the condition is `True then printing the position of the element.'
Implementing Linear Search
Below steps are required to implement the linear search.
Step1: Traverse the array using for loop.
Step2: In every iteration, compare the target value with the current value of the array
Step3: If the values match, return the current index of the array
Step4: If the values do not match, shift on to the next array element.
Step5: If no match is found, return -1

15. What is a Binary Search Tree?

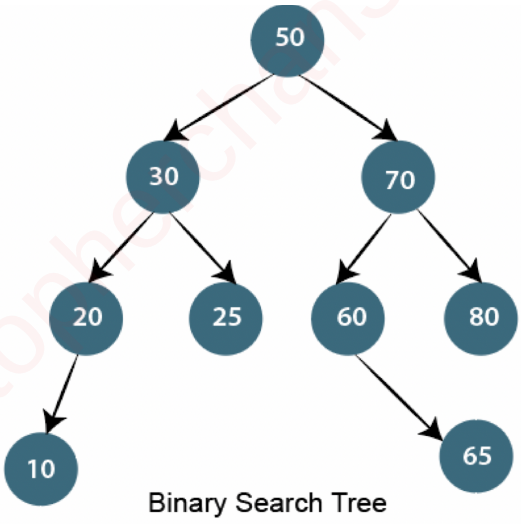
The binary search tree is a special type of data structure which has the following properties.

o Nodes which are less than root will be in the left subtree.

o Nodes which are greater than root (i.e., contains more value) will be right subtree.

o A binary search tree should not have duplicate nodes.

o Both sides subtree (i.e., left and right) also should be a binary search tree.



Binary Search Tree

16. Write an algorithm to insert a node in the Binary search tree?

Insert node operation is a smooth operation. You need to compare it with the root node and traverse left (if smaller) or right (if greater) according to the value of the node to be inserted.

Algorithm:

o Make the root node as the current node

o If the node to be inserted < root

o If it has left child, then traverse left

o If it does not have left child, insert node here

o If the node to be inserted > root

o If it has the right child, traverse right

o If it does not have the right child, insert node here.

17. How to count leaf nodes of the binary tree?

Algorithm-

Steps for counting the number of leaf nodes are:

o If the node is null (contains null values) then return 0.

o If encountered leaf node. Left is null and node Right is null then return 1.

o Recursively calculate the number of leaf nodes using

No. of leaf nodes= no of leaf nodes in left subtree + number of leaf nodes in the right subtree.

18. How to find all possible words in a board of characters (Boggle game)?

In the given dictionary, a process to do a lookup in the dictionary and an M x N board where every cell has a single character. Identify all possible words that can be formed by order of adjacent characters. Consider that we can move to any of the available 8 adjacent characters, but a word should not have multiple instances of the same cell.

Example:

```
1. dictionary[] = {"Java", "Point","Quiz"};
2. Array[][]   = {{'J', 'T', 'P',},
3.       {'U', 'A', 'A'},
4.       {'Q', 'S', 'V'}};
5. isWord(str): returns true if str is present in dictionary
6.     else false.
```

Output:

Following words of the dictionary are present

JAVA

19. Write an algorithm to insert a node in a link list?

Algorithm

o Check If the Linked list does not have any value then make the node as head and return it

o Check if the value of the node to be inserted is less than the value of the head node, then insert the node at the start and make it head.

o In a loop, find the appropriate node after which the input node is to be inserted. To find the just node start from the head, keep forwarding until you reach a node whose value is greater than the input node. The node just before is the appropriate node.

o Insert the node after the proper node found in step 3.

20. Explain how the encryption algorithm works?

Encryption is the technique of converting plaintext into a secret code format it is also called as "Ciphertext." To convert the text, the algorithm uses a string of bits called as "keys" for calculations. The larger the key, the higher the number of potential patterns for Encryption. Most of the algorithm use codes fixed blocks of input that have a length of about 64 to 128 bits, while some uses stream method for encryption.

21. What Are The Criteria Of Algorithm Analysis?

An algorithm is generally analyzed by two factors.

o Time complexity

o Space complexity

Time complexity deals with the quantification of the amount of time taken by a set of code or algorithm to process or run as a function of the amount of input. In other words, the time complexity is efficiency or how long a program function takes to process a given input.

Space complexity is the amount of memory used by the algorithm to execute and produce the result.

面试题

22. How to delete a node in a given link list? Write an algorithm and a program?

Write a function to delete a given node from a Singly Linked List. The function must follow the following constraints:

o The function must accept a pointer to the start node as the first argument and node to be deleted as the second argument, i.e., a pointer to head node is not global.

o The function should not return a pointer to the head node.

o The function should not accept pointer to pointer to head node.

We may assume that the Linked List never becomes empty.

Suppose the function name is delNode(). In a direct implementation, the function needs to adjust the head pointer when the node to be deleted the first node.

C program for deleting a node in Linked List

We will handle the case when the first node to be deleted then we copy the data of the next node to head and delete the next node. In other cases when a deleted node is not the head node can be handled generally by finding the previous node.

```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *next;
};

void delNode(struct Node *head, struct Node *n)
{
   if(head == n)
   {
      if(head->next == NULL)
      {
         printf("list can't be made empty because there is only one node. ");

         return;
      }
      head->data = head->next->data;
      n = head->next;
      head->next = head->next->next;
         free(n);
         return;
   }
      struct Node *prev = head;
   while(prev->next != NULL && prev->next != n)
      prev = prev->next;
   if(prev->next == NULL)
   {
      printf("\n This node is not present in  List");
      return;
   }
   prev->next = prev->next->next;
   free(n);
   return;
}
void push(struct Node **head_ref, int new_data)
{
   struct Node *new_node =
      (struct Node *)malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = *head_ref;
   *head_ref = new_node;
}
void printList(struct Node *head)
{
   while(head!=NULL)
   {
      printf("%d ",head->data);
      head=head->next;
   }
   printf("\n");
}
int main()
{
   struct Node *head = NULL;
   push(&head,3);
   push(&head,2);
   push(&head,6);
   push(&head,5);
   push(&head,11);

   push(&head,10);
   push(&head,15);
   push(&head,12);
   printf("Available Link list: ");
   printList(head);
   printf("\nDelete node %d: ", head->next->next->data);
   delNode(head, head->next->next);

   printf("\nUpdated  Linked List: ");
   printList(head);

   /* Let us delete the the first node */
   printf("\nDelete first node ");
   delNode(head, head);

   printf("\nUpdated Linked List: ");
   printList(head);

   getchar();
   return 0;
}
```

Output:

```
Available Link List: 12 15 10 11 5 6 2 3
Delete node 10:
Updated Linked List: 12 15 11 5 6 2 3
Delete first node
Updated Linked list: 15 11 5 6 2 3
```

面试题

23. Write a c program to merge a link list into another at an alternate position?

We have two linked lists, insert nodes of the second list into the first list at substitute positions of the first list.

Example

if first list is 1->2->3 and second is 12->10->2->4->6, the first list should become 1->12->2->10->17->3->2->4->6 and second list should become empty. The nodes of the second list should only be inserted when there are positions available.

Use of extra space is not allowed i.e., insertion must be done in a place.

Predictable time complexity is O(n) where n is number of nodes in first list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *next;
};
void push(struct Node ** head_ref, int new_data)
{
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
void printList(struct Node *head)
{
    struct Node *temp = head;
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void merge(struct Node *p, struct Node **q)
{
    struct Node *p_curr = p, *q_curr = *q;
    struct Node *p_next, *q_next;
    while (p_curr != NULL && q_curr != NULL)
    {
        p_next = p_curr->next;
        q_next = q_curr->next;
        q_curr->next = p_next;
        p_curr->next = q_curr;
        p_curr = p_next;
        q_curr = q_next;
    }

    *q = q_curr;
}
int main()
{
    struct Node *p = NULL, *q = NULL;
    push(&p, 3);
    push(&p, 2);
    push(&p, 1);
    printf("I Linked List:\n");
    printList(p);

    push(&q, 8);
    push(&q, 7);
    push(&q, 6);
    push(&q, 5);
    push(&q, 4);
    printf("II Linked List:\n");
    printList(q);

    merge(p, &q);

    printf("Updated I  Linked List:\n");
    printList(p);

    printf("Updated II Linked List:\n");
    printList(q);
        getchar();
    return 0;
}
```

Output:

```
I Linked List:
1 2 3
II Linked List:
4 5 6 7 8
Updated I Linked List:
1 4 2 5 3 6
Updated II Linked List:
7 8
```

24. What are the differences between stack and Queue?

Stack and Queue both are non-primitive data structure used for storing data elements and are based on some real-world equivalent.
Let's have a look at key differences based on the following parameters.

Working principle

The significant difference between stack and queue is that stack uses LIFO (Last in First Out) method to access and add data elements whereas Queue uses FIFO (First in first out) method to obtain data member.

Structure

In Stack, the same end is used to store and delete elements, but in Queue, one end is used for insertion, i.e., rear end and another end is used for deletion of elements.

Number of pointers used

Stack uses one pointer whereas Queue uses two pointers (in the simple case).

Operations performed

Stack operates as Push and pop while Queue operates as Enqueue and dequeuer.

Variants

Stack does not have variants while Queue has variants like a circular queue, Priority queue, doubly ended Queue.

Implementation

The stack is simpler while Queue is comparatively complex.

25. What is the difference between the Singly Linked List and Doubly Linked List data structure?

This is a traditional interview question on the data structure. The major difference between the singly linked list and the doubly linked list is the ability to traverse.
You cannot traverse back in a singly linked list because in it a node only points towards the next node and there is no pointer to the previous node.
On the other hand, the doubly linked list allows you to navigate in both directions in any linked list because it maintains two pointers towards the next and previous node.