\* Programming.

10. Give a one-line C expression to test whether a number is a power of 2.

**// This line will return a bool (c++) value**

**return x != 0 && x != INT\_MIN && (x & (x - 1)) == 0;**

11. Implement a smart pointer in C++.

**A smart pointer is an object that stores a pointer to a heap allocated object. If you use a smart pointer correctly, you no longer have to remember when to delete the new created memory.**

**Reference:**

**Crack the Code Interview**

**template <class X>**

**class Smart\_pointer**

**{**

**public:**

**// constructor**

**Smart\_pointer(X\* ptr)**

**{**

**ref = ptr;**

**ref\_count = (unsigned\*)malloc(sizeof(unsigned));**

**\*ref\_count = 1;**

**}**

**//copy constructor**

**Smart\_pointer(Smart\_pointer<X> & sptr)**

**{**

**ref = sptr.ref;**

**ref\_count = sptr.ref\_count;**

**++\*ref\_count ;**

**}**

**//destructor**

**~Smart\_pointer()**

**{**

**--\*ref\_count;**

**if(\*ref\_count == 0)**

**{**

**delete ref;**

**free(ref\_count);**

**ref = NULL;**

**ref\_count = NULL;**

**}**

**}**

**protected:**

**T\* ref;**

**unsigned\* ref\_count;**

**};**

12. Reverse a linked list from position m to n. Do it in-place and in one-pass.

**class ListNode {**

**public:**

**int value;**

**ListNode\* next;**

**ListNode(int v) : value(v), next(NULL) {}**

**};**

**// position is counted from 1**

**class Solution {**

**public:**

**ListNode \*reverseBetween(ListNode \*head, int m, int n) {**

**ListNode\* newHead = new ListNode(-1);**

**newHead->next = head;**

**ListNode\* prev = newHead;**

**for(auto i = 0 ; i < m - 1; i++){**

**prev = prev->next;**

**}**

**ListNode\* const reversedPrev = prev;**

**//position m**

**prev = prev->next;**

**ListNode\* cur = prev->next;**

**for(auto i = m ; i < n ; i++){**

**prev->next = cur->next;**

**cur->next = reversedPrev->next;**

**reversedPrev->next = cur;**

**cur = prev->next;**

**}**

**return newHead->next;**

**}**

**};**

13. Implement a program to find out whether there exist M days within the last N(N >= M) trading days that the average closing price of these M days is at most P. Assume we have collected the history of the closing prices of the last N trading days for a stock. Requirements:

Inputs are positive integer M and N, M <= N; An array of N float elements containing the closing prices of the last N trading days; And a float P. Please design and implement the program in C, C++, Java or Python to produce the answer in most time/space efficient way.

**\* If the M elements have to be continuous, this will be a moving average problem. Time complexity is O(N). Space complexity is O(1).**

**public class Solution {**

**// return the starting index if this M days exsit, otherwise return -1**

**public static int checkaverage(int[] prices, int N, int M, double P)**

**{**

**double target\_sum = P \* M;**

**int sum = 0;**

**boolean foundit = false;**

**int i =0;**

**for(; i < M; i++)**

**{**

**sum += prices[i];**

**}**

**while(i < N)**

**{**

**if(sum <= target\_sum)**

**{**

**foundit = true;**

**break;**

**}**

**sum = sum + prices[i] - prices[i - M];**

**i++;**

**}**

**return foundit? i - M:-1;**

**}**

**}**

**\* If the M elements do not have to be continuous. Time complexity is O(NlogN). Space complexity is O(M)**

**a. use a max heap to save the first M elements**

**b. everytime compare the root of the max heap with the new element, keep the smaller one in the heap and remove the larger one, until the average of the max heap is less than or equal to P**

14. Implement a string indexOf method that returns index of matching string.

**// Sundy algorithm: faster than KMP and BM algorithm. Linear time complexity O(M+N) , in java**

**public static HashMap<Character, Integer> buildindex(String pattern)**

**{**

**HashMap<Character, Integer> position = new HashMap<Character, Integer>();**

**for(int i = pattern.size() – 1; i >=0; i--)**

**{**

**if (!position.contaisKey(pattern.charAt(i)))**

**position.put(pattern.charAt(i), i);**

**}**

**return position;**

**}**

**public static int indexOf(String pattern, String text)**

**{**

**HashMap<Character, Integer> position = buildindex(pattern);**

**if(pattern == null || text == null) return -1;**

**pLen = pattern.size();**

**tLen = text.size();**

**if(tLen < pLen) return -1;**

**int I = 1;**

**int j = 1;**

**while(i < pLen && j < tLen)**

**{**

**if(pattern.charAt(i) == text.charAt(j))**

**{**

**i++;**

**j++;**

**}**

**else**

**{**

**char check = text.charAt(j + pLen);**

**int pos;**

**if(position.containsKey(check))**

**pos = position.get(check);**

**else**

**pos = -1;**

**i = 0;**

**j += pLen – pos;**

**}**

**}**

**if(i == pLen) return j-i;**

**else return -1;**

**}**

**// Brute force method. O(N2)**

**int indexOf (string s1, string s2) {**

**if(s1 == "" || s2 == "") { // if s1 is null or s2 is null, return -1**

**return -1;**

**}**

**int i, j;**

**for(i = 0; i <= s1.length()- s2.length(); i++) { // try every possible start index in s1**

**for(j = 0; j < s2.length(); j++) { // for every letter in s2**

**if(s1[i+j] != s2[j]) {**

**break;**

**}**

**}**

**if (j == s2.length()) {**

**return i;**

**}**

**}**

**return -1;**

**}**

15. Write a function to calculate exp(x).

**// C++**

**double exp(double x) { // x is the power exponent, delta is the practical limit for 0**

**double term = 1; // first term**

**double sum = 0; // initial value**

**for (int i = 1; sum != sum + term; ++i) {**

**sum = sum + term;**

**term = term\*x/i;**

**}**

**return exp;**

**}**

16. Given streaming data, design an algorithm to get approximate median of all previous data, use constant memory.

**// c++**

**class Solution { // two heaps, min heap and max heap**

**public:**

**vector<int> median(vector<int> &nums) {**

**vector<int> result;**

**if (nums.size() == 0) {**

**return result;**

**}**

**int median = nums[0];**

**priority\_queue<int> maxHeap, minHeap;**

**result.push\_back(median);**

**for (int i = 1; i < nums.size(); i++) {**

**if (nums[i] < median) {**

**maxHeap.push(nums[i]);**

**} else {**

**minHeap.push(nums[i]);**

**}**

**if (maxHeap.size() > minHeap.size()) {**

**minHeap.push(median);**

**median = maxHeap.top();**

**maxHeap.pop();**

**} else if (maxHeap.size() + 1 < minHeap.size()) {**

**maxHeap.push(median);**

**median = minHeap.top();**

**minHeap.pop();**

**}**

**result.push\_back(median);**

**}**

**return result;**

**}**

**};**

**//java**

**public class Solution {**

**public static int getMedian(int new\_elem,int initialmedian, PriorityQueue<Integer> left, PriorityQueue<Integer> right)**

**{**

**if(left.size() == right.size())**

**{**

**int result;**

**if(new\_elem < initialmedian)**

**{**

**left.add(new\_elem);**

**result = left.peek();**

**}**

**else**

**{**

**right.add(new\_elem);**

**result = right.peek();**

**}**

**return result;**

**}**

**else if(left.size() > right.size())**

**{**

**if(new\_elem < initialmedian)**

**{**

**right.add(left.poll());**

**left.add(new\_elem);**

**}**

**else**

**{**

**right.add(new\_elem);**

**}**

**return (left.peek() + right.peek()) / 2;**

**}**

**else**

**{**

**if(new\_elem < initialmedian)**

**{**

**left.add(new\_elem);**

**}**

**else**

**{**

**left.add(right.poll());**

**right.add(new\_elem);**

**}**

**return (left.peek() + right.peek()) / 2;**

**}**

**}**

**public static void main(String[] args)**

**{**

**int initialmedian = 0;**

**int[] prices = {3,6,7,9,3,2,4,8,9};**

**PriorityQueue<Integer> left = new PriorityQueue<Integer>((x, y) -> y - x);**

**PriorityQueue<Integer> right = new PriorityQueue<Integer>();**

**for(int i = 0; i < prices.length; i++)**

**{**

**initialmedian = getMedian(prices[i], initialmedian, left, right);**

**System.out.println(initialmedian);**

**}**

**}**

**}**

17. Say you have an array for which the i-th element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e. buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (i.e. you must sell the stock before you buy again).

**class Solution {**

**public:**

**int maxProfit(vector<int> &prices) {**

**int len = prices.size();**

**int profit = 0;**

**for(int i = 0; i < len; i++)**

**{**

**if (prices[i + 1] > prices[i])**

**profit += prices[i+1] – price[i];**

**}**

**return profit;**

**}**

**};**

Project Examples:

[**https://www.kaggle.com/c/AlgorithmicTradingChallenge**](https://www.kaggle.com/c/AlgorithmicTradingChallenge)

[**https://www.kaggle.com/c/battlefin-s-big-data-combine-forecasting-challenge**](https://www.kaggle.com/c/battlefin-s-big-data-combine-forecasting-challenge)

[**https://www.kaggle.com/c/loan-default-prediction**](https://www.kaggle.com/c/loan-default-prediction)