**1.**

**#include <stdio.h>**

**Int main() {**

**Int n, m;**

**Scanf(“%d %d”, &n, &m);**

**Int arr[n];**

**For (int i = 0; i < n; i++) {**

**Scanf(“%d”, &arr[i]);**

**}**

**Long long count = 0;**

**Int start = 0, used\_ops = 0;**

**For (int end = 0; end < n; end++) {**

**If (end > 0 && arr[end] < arr[end – 1]) {**

**Used\_ops += arr[end – 1] – arr[end] + 1;**

**}**

**While (used\_ops > m) {**

**If (start + 1 < n && arr[start] > arr[start + 1]) {**

**Used\_ops -= arr[start] – arr[start + 1] + 1;**

**}**

**Start++;**

**}**

**Count += end – start + 1;**

**}**

**Printf(“%lld\n”, count);**

**Return 0;**

**}**

**2.**

**#include <stdio.h>**

**#include <math.h>**

**Typedef struct {**

**Double x, y;**

**} Point;**

**Int main() {**

**Int n;**

**Double X;**

**Scanf(“%d %lf”, &n, &X);**

**Point points[n];**

**For (int i = 0; i < n; i++) {**

**Scanf(“%lf %lf”, &points[i].x, &points[i].y);**

**}**

**Double result[n];**

**For (int i = 0; i < n; i++) {**

**Result[i] = 0.0;**

**}**

**For (int i = 0; i < n; i++) {**

**Double left = (i == 0) ? 0 : (points[i].x + points[i – 1].x) / 2.0;**

**Double right = (i == n – 1) ? X : (points[i].x + points[i + 1].x) / 2.0;**

**If (left < 0) left = 0;**

**If (right > X) right = X;**

**Result[i] += right – left;**

**}**

**For (int i = 0; i < n; i++) {**

**Printf(“%.6lf\n”, result[i]);**

**}**

**Return 0;**

**}**

**3.**

**#include <stdio.h>**

**#include <limits.h>**

**Int canMinimize(int arr[], int n, int k, int maxDiff) {**

**Int removed = 0;**

**For (int i = 1; i < n; i++) {**

**If (arr[i] – arr[i – 1] > maxDiff) {**

**Removed++;**

**If (removed > k) return 0;**

**}**

**}**

**Return 1;**

**}**

**Int main() {**

**Int n, k;**

**Scanf(“%d %d”, &n, &k);**

**Int arr[n];**

**For (int i = 0; i < n; i++) {**

**Scanf(“%d”, &arr[i]);**

**}**

**Int left = 0, right = arr[n – 1] – arr[0], result = INT\_MAX;**

**While (left <= right) {**

**Int mid = left + (right – left) / 2;**

**If (canMinimize(arr, n, k, mid)) {**

**Result = mid;**

**Right = mid – 1;**

**} else {**

**Left = mid + 1;**

**}**

**}**

**Printf(“%d\n”, result);**

**Return 0;**

**}**

**4.**

**#include <stdio.h>**

**Void swap(int\* a, int\* b) {**

**Int temp = \*a;**

**\*a = \*b;**

**\*b = temp;**

**}**

**Int main() {**

**Int n, k;**

**Scanf(“%d %d”, &n, &k);**

**Int arr[n];**

**For (int i = 0; i < n; i++) {**

**Scanf(“%d”, &arr[i]);**

**}**

**For (int i = 0; i < n; i++) {**

**Int min\_idx = i;**

**For (int j = i + 1; j < n && j <= i + k; j++) {**

**If (arr[j] < arr[min\_idx]) {**

**Min\_idx = j;**

**}**

**}**

**If (min\_idx != i) {**

**For (int j = min\_idx; j > i; j--) {**

**Swap(&arr[j], &arr[j – 1]);**

**k--;**

**if (k == 0) break;**

**}**

**}**

**If (k == 0) break;**

**}**

**For (int i = 0; i < n; i++) {**

**Printf(“%d “, arr[i]);**

**}**

**Printf(“\n”);**

**Return 0;**

**}**

**5. 4th and 5th program are same**

**6.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#define MAX\_N 100005**

**Int main() {**

**Int n, k, m, s;**

**Scanf(“%d %d %d %d”, &n, &k, &m, &s);**

**s--;**

**int forbidden[MAX\_N] = {0};**

**for (int i = 0; i < m; i++) {**

**int pos;**

**scanf(“%d”, &pos);**

**forbidden[pos – 1] = 1;**

**}**

**Int dist[MAX\_N];**

**For (int i = 0; i < n; i++) dist[i] = -1;**

**Dist[s] = 0;**

**Int queue[MAX\_N], front = 0, rear = 0;**

**Queue[rear++] = s;**

**While (front < rear) {**

**Int curr = queue[front++];**

**Int currDist = dist[curr];**

**For (int i = -k + 1; i < k; i++) {**

**Int start = curr + i, end = curr + i + k – 1;**

**If (start < 0 || end >= n) continue;**

**Int target = start + end – curr;**

**If (target < 0 || target >= n || forbidden[target]) continue;**

**If (dist[target] == -1) {**

**Dist[target] = currDist + 1;**

**Queue[rear++] = target;**

**}**

**}**

**}**

**For (int i = 0; i < n; i++) printf(“%d “, dist[i]);**

**Printf(“\n”);**

**Return 0;**

**}**

**7.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <math.h>**

**Typedef struct {**

**Long long left, right, center;**

**} Platform;**

**Int comparePlatforms(const void\* a, const void\* b) {**

**Platform\* p1 = (Platform\*)a;**

**Platform\* p2 = (Platform\*)b;**

**Return (p1->center > p2->center) – (p1->center < p2->center);**

**}**

**Int compareBalls(const void\* a, const void\* b) {**

**Long long\* b1 = (long long\*)a;**

**Long long\* b2 = (long long\*)b;**

**Return (\*b1 > \*b2) – (\*b1 < \*b2);**

**}**

**Long long minCost(long long ball, Platform platform) {**

**If (ball < platform.left) return platform.left – ball;**

**If (ball > platform.right) return ball – platform.right;**

**Return 0;**

**}**

**Int main() {**

**Int n, m;**

**Scanf(“%d %d”, &n, &m);**

**Platform platforms[n];**

**For (int i = 0; i < n; i++) {**

**Scanf(“%lld %lld”, &platforms[i].left, &platforms[i].right);**

**Platforms[i].center = (platforms[i].left + platforms[i].right) / 2;**

**}**

**Long long balls[m];**

**For (int i = 0; i < m; i++) {**

**Scanf(“%lld”, &balls[i]);**

**}**

**Qsort(platforms, n, sizeof(Platform), comparePlatforms);**

**Qsort(balls, m, sizeof(long long), compareBalls);**

**Long long totalCost = 0;**

**For (int i = 0; i < n; i++) {**

**Platform p = platforms[i];**

**Long long\* closest = (long long\*)bsearch(&p.center, balls, m, sizeof(long long), compareBalls);**

**Long long cost = LLONG\_MAX;**

**If (closest != NULL) {**

**Cost = minCost(\*closest, p);**

**}**

**Int idx = (closest != NULL) ? (closest – balls) : -1;**

**If (idx > 0) cost = fmin(cost, minCost(balls[idx – 1], p));**

**If (idx + 1 < m) cost = fmin(cost, minCost(balls[idx + 1], p));**

**totalCost += cost;**

**}**

**Printf(“%lld\n”, totalCost);**

**Return 0;**

**}**

**8.**

**#include <stdio.h>**

**#include <string.h>**

**#include <stdlib.h>**

**#define MAXN 200005**

**Char s[MAXN];**

**Int n, m;**

**Int is\_palindrome(int i, int j) {**

**While(i < j) {**

**If(s[i] != s[j]) return 0;**

**I++, j--;**

**}**

**Return 1;**

**}**

**Void mod1(int i, int j, int k) {**

**Int len = j – i + 1;**

**Char temp[MAXN];**

**Strncpy(temp, s + i, len);**

**Memmove(s + i, s + j + 1, n – j – 1);**

**S[n – len] = 0;**

**Strncat(s, temp, len);**

**}**

**Void mod2(int i, int j) {**

**While(i < j) {**

**Char temp = s[i];**

**S[i] = s[j];**

**S[j] = temp;**

**I++, j--;**

**}**

**}**

**Void mod3(int i, char c) {**

**Memmove(s + i + 1, s + i, n – i);**

**S[i] = c;**

**N++;**

**}**

**Int main() {**

**Scanf(“%d %d”, &n, &m);**

**Scanf(“%s”, s);**

**For(int q = 0; q < m; q++) {**

**Char type;**

**Int i, j, k;**

**Char c;**

**Scanf(“ %c”, &type);**

**If(type == ‘Q’) {**

**Scanf(“%d %d”, &i, &j);**

**Printf(is\_palindrome(i – 1, j – 1) ? “YES\n” : “NO\n”);**

**} else if(type == ‘M’) {**

**Scanf(“%d %d”, &i, &j);**

**If(i == 1) {**

**Scanf(“%d”, &k);**

**Mod1(i – 1, j – 1, k – 1);**

**} else if(i == 2) {**

**Mod2(i – 1, j – 1);**

**} else if(i == 3) {**

**Scanf(“ %c”, &c);**

**Mod3(i – 1, c);**

**}**

**}**

**}**

**Return 0;**

**}**

**9.**

**#include <stdio.h>**

**#include <string.h>**

**#include <ctype.h>**

**#define MAX\_N 105**

**#define MAX\_LEN 1000**

**Int order\_map[256];**

**Int compare(const void \*a, const void \*b) {**

**Char \*str1 = \*(char \*\*)a;**

**Char \*str2 = \*(char \*\*)b;**

**Int i = 0;**

**While (str1[i] && str2[i]) {**

**If (order\_map[str1[i]] != order\_map[str2[i]]) {**

**Return order\_map[str1[i]] – order\_map[str2[i]];**

**}**

**I++;**

**}**

**Return strlen(str1) – strlen(str2);**

**}**

**Int main() {**

**Char permutation[27];**

**Int N;**

**Scanf(“%s”, permutation);**

**For (int i = 0; i < 26; i++) {**

**Order\_map[permutation[i]] = i;**

**Order\_map[permutation[i] – 32] = i + 26;**

**}**

**Scanf(“%d”, &N);**

**Char \*words[MAX\_N];**

**For (int i = 0; i < N; i++) {**

**Words[i] = (char \*)malloc(MAX\_LEN \* sizeof(char));**

**Scanf(“%s”, words[i]);**

**}**

**Qsort(words, N, sizeof(char \*), compare);**

**For (int i = 0; i < N; i++) {**

**Printf(“%s\n”, words[i]);**

**Free(words[i]);**

**}**

**Return 0;**

**}**

**10.**

**#include <stdio.h>**

**#include <string.h>**

**#include <stdlib.h>**

**#define ALPHABET\_SIZE 26**

**#define MAX\_WORDS 100005**

**Char normalized[MAX\_WORDS][ALPHABET\_SIZE + 1];**

**Void normalize\_word(char \*word, char \*normalized\_word) {**

**Int offset = word[0] – ‘a’;**

**For (int i = 0; word[i] != ‘\0’; i++) {**

**Normalized\_word[i] = ((word[i] – ‘a’ – offset + ALPHABET\_SIZE) % ALPHABET\_SIZE) + ‘a’;**

**}**

**Normalized\_word[strlen(word)] = ‘\0’;**

**}**

**Int compare(const void \*a, const void \*b) {**

**Return strcmp((const char \*)a, (const char \*)b);**

**}**

**Int main() {**

**Int N;**

**Scanf(“%d”, &N);**

**Char word[ALPHABET\_SIZE + 1];**

**For (int i = 0; i < N; i++) {**

**Scanf(“%s”, word);**

**Normalize\_word(word, normalized[i]);**

**}**

**Qsort(normalized, N, sizeof(normalized[0]), compare);**

**Int result[MAX\_WORDS] = {0};**

**For (int i = 1; i < N; i++) {**

**If (strcmp(normalized[i], normalized[i – 1]) == 0) {**

**Result[i] = 1;**

**Result[i – 1] = 1;**

**}**

**}**

**For (int i = 0; i < N; i++) {**

**Printf(“%d\n”, result[i]);**

**}**

**Return 0;**

**}**

**12.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_N 100005**

**Int cmp\_desc(const void \*a, const void \*b) {**

**Return \*(int \*)b - \*(int \*)a;**

**}**

**Int cmp\_asc(const void \*a, const void \*b) {**

**Return \*(int \*)a - \*(int \*)b;**

**}**

**Int main() {**

**Int N, M;**

**Scanf(“%d %d”, &N, &M);**

**Int alex[MAX\_N], ben[MAX\_N];**

**For (int i = 0; i < N; i++) {**

**Scanf(“%d”, &alex[i]);**

**}**

**For (int i = 0; i < M; i++) {**

**Scanf(“%d”, &ben[i]);**

**}**

**Qsort(alex, N, sizeof(int), cmp\_desc);**

**Qsort(ben, M, sizeof(int), cmp\_asc);**

**Int alex\_index = 0, ben\_index = 0;**

**Long long unused\_sum = 0;**

**While (alex\_index < N && ben\_index < M) {**

**If (alex[alex\_index] > ben[ben\_index]) {**

**Ben\_index++;**

**} else {**

**Unused\_sum += alex[alex\_index];**

**}**

**Alex\_index++;**

**}**

**For (int i = alex\_index; i < N; i++) {**

**Unused\_sum += alex[i];**

**}**

**If (ben\_index < M) {**

**Printf(“-1\n”);**

**} else {**

**Printf(“%lld\n”, unused\_sum);**

**}**

**Return 0;**

**}**

**13.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define ll long long**

**Int cmp(const void \*a, const void \*b) {**

**Return \*(int \*)a - \*(int \*)b;**

**}**

**Int main() {**

**Int N, R, B;**

**Scanf(“%d %d %d”, &N, &R, &B);**

**Int red[R], blue[B];**

**For (int i = 0; i < R; i++) {**

**Scanf(“%d”, &red[i]);**

**}**

**For (int i = 0; i < B; i++) {**

**Scanf(“%d”, &blue[i]);**

**}**

**Qsort(blue, B, sizeof(int), cmp);**

**Ll total\_distance = 0;**

**For (int i = 0; i < R; i++) {**

**Int left = 0, right = B – 1, closest\_dist = N;**

**While (left <= right) {**

**Int mid = (left + right) / 2;**

**Int dist = abs(red[i] – blue[mid]);**

**If (dist < closest\_dist) {**

**Closest\_dist = dist;**

**}**

**If (blue[mid] < red[i]) {**

**Left = mid + 1;**

**} else {**

**Right = mid – 1;**

**}**

**}**

**Total\_distance += closest\_dist;**

**}**

**Printf(“%lld\n”, total\_distance);**

**Return 0;**

**}**

**14.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <stdbool.h>**

**#define MOD 1000000007**

**Typedef struct { int \*data, size, cap; } Vector;**

**Void init(Vector \*v, int cap) { v->data = malloc(cap \* sizeof(int)); v->size = 0; v->cap = cap; }**

**Void push(Vector \*v, int val) { if (v->size == v->cap) v->data = realloc(v->data, (v->cap \*= 2) \* sizeof(int)); v->data[v->size++] = val; }**

**Void free\_vec(Vector \*v) { free(v->data); }**

**Vector \*adj;**

**Int \*dist, \*queue;**

**Bool \*vis;**

**Bool is\_connected(int n) {**

**Int front = 0, rear = 0, count = 1;**

**Vis[1] = true, queue[rear++] = 1;**

**While (front < rear) {**

**Int u = queue[front++];**

**For (int i = 0; i < adj[u].size; i++) {**

**Int v = adj[u].data[i];**

**If (!vis[v]) vis[v] = true, queue[rear++] = v, count++;**

**}**

**}**

**Return count == n;**

**}**

**Long long count\_graphs(int n) {**

**Vector \*level = malloc(n \* sizeof(Vector));**

**For (int i = 0; i < n; i++) init(&level[i], 2);**

**For (int i = 1; i <= n; i++) {**

**If (dist[i] >= n) { for (int j = 0; j < n; j++) free\_vec(&level[j]); free(level); return 0; }**

**Push(&level[dist[i]], i);**

**}**

**If (!is\_connected(n)) { for (int j = 0; j < n; j++) free\_vec(&level[j]); free(level); return 0; }**

**Long long ways = 1;**

**For (int i = 1; i < n; i++) {**

**If (!level[i].size) continue;**

**Int count = 0;**

**For (int j = 0; j < level[i].size; j++) {**

**Int u = level[i].data[j];**

**For (int k = 0; k < adj[u].size; k++) {**

**If (dist[adj[u].data[k]] == i – 1) count++;**

**}**

**}**

**Ways = (ways \* count) % MOD;**

**}**

**For (int i = 0; i < n; i++) free\_vec(&level[i]);**

**Free(level);**

**Return ways;**

**}**

**Int main() {**

**Int n, m;**

**Scanf(“%d %d”, &n, &m);**

**Dist = malloc((n + 1) \* sizeof(int));**

**Vis = calloc(n + 1, sizeof(bool));**

**Queue = malloc(n \* sizeof(int));**

**Adj = malloc((n + 1) \* sizeof(Vector));**

**For (int i = 1; i <= n; i++) init(&adj[i], 2), scanf(“%d”, &dist[i]);**

**For (int i = 0, u, v; i < m; i++) scanf(“%d %d”, &u, &v), push(&adj[u], v), push(&adj[v], u);**

**Printf(“%lld\n”, count\_graphs(n));**

**Free(dist), free(vis), free(queue);**

**For (int i = 1; i <= n; i++) free\_vec(&adj[i]);**

**Free(adj);**

**Return 0;**

**}**