2021 提高组

1.在 Linux 系统终端中，用于列出当前目录下所含的文件和子目录的命令为（ ）。
A. ls　　B. cd　　C. cp　　D. all
2.二进制数 001010102 和 000101102 的和为（）。
A. 001111002　　B. 010000002　　C. 001111002　　D. 010000102
3.在程序运行过程中，如果递归调用的层数过多，可能会由于（ ）引发错误。
A. 系统分配的栈空间溢出　　B. 系统分配的队列空间溢出
C. 系统分配的链表空间溢出　　D. 系统分配的堆空间溢出
4.以下排序方法中，（ ）是不稳定的。
A. 插入排序　　B. 冒泡排序　　C. 堆排序　　D. 归并排序
5.以比较为基本运算，对于 2n 个数，同时找到最大值和最小值，最坏情况下需要的最小的比较次数为（ ）。
A. 4n−2　　B. 3n+1　　C. 3n−2　　D. 2n+1
6.现有一个地址区间为 0~10 的哈希表，对于出现冲突情况，会往后找第一个空的地址存储（到 10 冲突了就从 0 开始往后），现在要依次存储 (0,1,2，3,4,5,6,7)，哈希函数为 h(x)=x2 mod11。请问 77 存储在哈希表哪个地址中（ ）。
A. 5　　B. 6　　C. 7　　D. 8
7.G 是一个非连通简单无向图（没有自环和重边），共有 36 条边，则该图至少有（ ）个点。
A. 8　　B. 9　　C. 10　　D. 11
8.令根结点的高度为 1，则一棵含有 2021 个结点的二叉树的高度至少为（ ）。
A. 10　　B. 11　　C. 12　　D. 2021
9.前序遍历和中序遍历相同的二叉树为且仅为（ ）。
A. 只有 1 个点的二叉树
B. 根结点没有左子树的二叉树
C. 非叶子结点只有左子树的二叉树
D. 非叶子结点只有右子树的二叉树
10.定义一种字符串操作为交换相邻两个字符。将 DACFEB 变为 ABCDEF 最少需要（ ）次上述操作。
A. 7　　B. 8　　C. 9　　D. 6
11.有如下递归代码

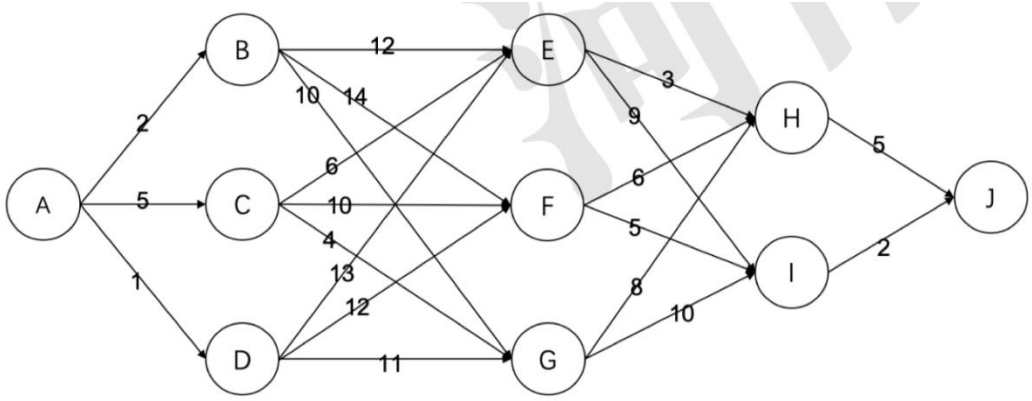
```
solve(t, n):
  if t=1 return 1
  else return 5*solve(t-1,n) mod n
```

则 solve(23,23) 的结果为（ ）。
A. 1　　B. 7　　C. 12　　D. 22
12.斐波那契数列的定义为： $F_1=1$，$F_2=1$，$F_n=F_{n-1}+F_{n-2}(n\geq 3)$。现在用如下程序来计算斐波那契数列的第 n 项，其时间复杂度为（ ）。

```
F(n):
  if n<=2 return 1
  else return F(n-1) + F(n-2)
```

A. $O(n)$　　B. $O(n^2)$　　C. $O(2^n)$　　D. $O(n \log n)$
13.有 8 个苹果从左到右排成一排，你要从中挑选至少一个苹果，并且不能同时挑选相邻的两个苹果，一共有（ ）种方案。
A. 36　　B. 48　　C. 54　　D. 64
14.设一个三位数 n=abc，a，b，c 均为 1~9 之间的整数，若以 a、 b、 c 作为三角形的三条边可以构成等腰三角形（包括等边），则这样的 n 有（ ）个。
A. 81　　B. 120　　C. 165　　D. 216
15.有如下的有向图，节点为 A,B,…,J，其中每条边的长度都标在图中。则节点 A 到节点

J 的最短路径长度为（ ）。



A. 16　　B. 19　　C. 20　　D. 22

二、阅读程序（程序输入不超过数组或字符串定义的范围；判断题正确填 √ ，错误填 × ；除特殊说明外，判断题 1.5 分，选择题 3 分，共计 40 分）

16.（本题共 12 分）

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std ; .
4 const double r = acos(0.5);
5 int a1, b1, c1, d1;
6 int a2, b2, C2, d2;
7 inline int sq(const int x) { return x * x; }
8 inline int cu(const int x) { return x*x*x;}
9 int main()
10 {
11  cout . flags(ios: :fixed);
12  cout.precision(4);
13  cin>>a1>>b1>>c1>>d1;
14  cin>>a2>>b2>>c2>>d2;
15  int t=sq(a1-a2)+sq(b1-b2)+sq(c1-c2);
16  if (t <= sq(d2 - d1)) cout << cu(min(d1, d2)) *r*4;
17  else if (t >= sq(d2 + d1)) cout << 0;
18  else {
19    double x = d1 - (sq(d1) - sq(d2) + t) / sqrt(t) / 2;
20    double y = d2 - (sq(d2) - sq(d1) + t) / sqrt(t) / 2;
21    cout << (x*x*(3*d1-x)+y*y*(3*d2-y))*r;
22   }
23  cout << endl;
24  return 0;
25 }
```

假设输入的所有数的绝对值都不超过 1000 ，完成下面的判断题和单选题：

●判断题

(1)将第 15 行中 t 的类型声明从 int 改为 double，不会影响程序运行的结果。（　　）

(2)将第 19、20 行中的/sqrt(t)/2 替换为/2/sqrt(t),不会影响程序运行的结果。（　　）

(3)将第 21 行中的 x*x 改成 sq(x)、y*y 改成 sq(y)，不会影响程序运行的结果。（　　）

(4)(2 分)当输入为 0 0 0 1 1 0 0 1 时，输出为 1.3090（　　）

●单选题

(5)当输入为 1 1 1 1 1 1 1 2 时，输出为（ ）。

A. 3.1416　　B. 6.2832　　C. 4.7124　　D. 4.1888

(6)(2.5 分)这段代码的含义为（ ）。

A. 求圆的面积并　　B. 求球的体积并

C. 求球的体积交　　D. 求椭球的体积并

17.（本题共 13.5 分）

```
#include <algorithm>
#include <iostream>
using namespace std;
int n, a[1005];
struct Node
{
   int h,j,m,W;
    Node(const int_ h, const int_ j, const int_ m,  const int_ w):
      h(_ h), j(_ _j), m(_ m), W(. w)
      {}
Node operator+(const Node &o) const
 {
  return Node(
        max(h, W + o.h),
        max(max(j, o.j), m + o.h),
        max(m + 0.w, o.m),
        w + o.w);
  }
 };
 Node solve1(int h, int m)
{
  if(h> m)
    return Node(-1, -1, -1, -1);
  if(h==m)
    return Node(max(a[h], 0), max(a[h], 0), max(a[h], 0), a[h]);
  int j =(h+ m)>>1;
    return solve1(h, j) + solve1(j + 1, m);
 }
 int solve2(int h, int m)
  {
   if(h> m)
     return -1;
   if(h==m)
     return max(a[h], 0);
   int j=(h+m)>>1;
   int wh=0,wm=0;
   int wht=0, wmt=0;
   for(int i=j;i>=h;i--){
     wht += a[i];
     wh = max(wh, wht);
   }
  for(int i=j+1;i<=m;i++){
    wmt += a[i];
    wm = max(wm, wmt);
  }
   return max(max(solve2(h, j), solve2(j + 1, m)), wh + wm);
}
```

```
48int main()
49{
50  cin>>n;
51   for(int i=1;i<=n;i++)cin>>a[i];
52   cout << solve1(1, n).j << endl;
53   cout << solve2(1, n) << endl;
54   return 0;
55 }
```

假设输入的所有数的绝对值都不超过 1000 ，完成下面的判断题和单选题：

●判断题

(1)程序总是会正常执行并输出两行两个相等的数。(　　)

(2)第 23 行与第 32 行分别有可能执行两次及以上。 (　　)

(3)当输入为 5 -10 11 -9 5 -7 时，输出的第二行为 7。(　　)

●单选题

(4)solve1(1, n) 的时间复杂度为（ ）。

A. O(log n)　　B. O(n)　　C. O(n log n)　　D. $O(n^2)$

(5)solve2(1, n) 的时间复杂度为（ ）。

A. O(log n)　　B. O(n)　　C. O(n log n)　　D. $O(n^2)$

(6)当输入为 10 -3 2 10 0 -8 9 -4 -5 9 4 时，输出的第一行为（ ）。

A. 13　　B. 17　　C. 24　　D. 12

18.（本题共 14.5 分）

```
01 #include <iostream>
02 #include <string>
03 using namespace std;
04
05 char base[64];
06 char table[256];
07
08 void init()
09 {
10    for (int i = 0; i < 26; i++) base[i] = 'A' + i;
11    for (int i = 0; i < 26; i++) base[26 + i] = 'a' + i;
12    for (int i = 0; i < 10; i++) base[52 + i] = '0' + i;
13    base[62] = '+', base[63] = '/';
14
15    for (int i = 0; i < 256; i++) table[i] = 0xff;
16    for (int i = 0; i < 64; i++) table[base[i]] = i;
17    table['='] = 0;
18 }
19
20 string encode(string str)
21 {
22    string ret;
23    int i;
24    for (i = 0; i + 3 <= str.size(); i += 3) {
25        ret += base[str[i] >> 2];
26        ret += base[(str[i] & 0x03) << 4 | str[i + 1] >> 4];
27        ret += base[(str[i + 1] & 0x0f) << 2 | str[i + 2] >> 6];
28        ret += base[str[i + 2] & 0x3f];
29    }
```

```
30    if (i < str.size()) {
31        ret += base[str[i] >> 2];
32        if (i + 1 == str.size()) {
33            ret += base[(str[i] & 0x03) << 4];
34           ret += "==";
35        }
36        else {
37           ret += base[(str[i] & 0x03) << 4 | str[i + 1] >> 4];
38           ret += base[(str[i + 1] & 0x0f) << 2];
39           ret += "=";
40        }
41    }
42    return ret;
43 }
44
45 string decode(string str)
46 {
47    string ret;
48    int i;
49    for (i = 0; i < str.size(); i += 4) {
50        ret += table[str[i]] << 2 | table[str[i + 1]] >> 4;
51         if (str[i + 2] != '=')
52            ret += (table[str[i + 1]] & 0x0f) << 4 | table[str[i +2]] >> 2;
53         if (str[i + 3] != '=')
54            ret += table[str[i + 2]] << 6 | table[str[i + 3]];
55     }
56    return ret;
57 }
58
59 int main()
60 {
61    init();
62    cout << int(table[0]) << endl;
63
64    int opt;
65    string str;
66    cin >> opt >> str;
67    cout << (opt ? decode(str) : encode(str)) << endl;
68    return 0;
69 }
```

假设输入总是合法的（一个整数和一个不含空白字符的字符串，用空格隔开），完成下面的判断题和单选题：

●判断题

(1)程序总是先输出一行一个整数，再输出一行一个字符串。(　　)

(2)对于任意不含空白字符的字符串 str1，先执行程序输入 0 str1，得到输出的第二行记为 str2；再执行程序输入 1 str2，输出的第二行必为 str1。(　　)

(3)当输入为 1 SGVsbG93b3JsZA==时，输出的第二行为 HelloWorld。(　　)

●单选题

(4)设输入字符串长度为 n，encode 函数的时间复杂度为（ ）。

A. O(√n)　　B. O(n)　　C. O(n log n)　　D. $O(n^2)$

(5)输出的第一行为（ ）。

A. 0xff　　B. 255　　C. 0xFF　　D. -1

(6)(4 分) 当输入为 0 CSP2021csp 时，输出的第二行为（ ）。

A. Q1NQMjAyMWNzcAv=　　B. Q1NQMjAyMGNzcA==

C. Q1NQMjAyMGNzcAv=　　D. Q1NQMjAyMWNzcA==

三、完善程序（单选题，每小题 3 分，共计 30 分）

19.(魔法数字)小 H 的魔法数字是 4。给定 n，他希望用若干个 4 进行若干次加法、减法和整除运算得到 n。但由于小 H 计算能力有限，计算过程中只能出现不超过 M = 10000 的正整数。求至少可能用到多少个 4。本题共 12 分

例如，当 n=2 时，有 2= (4+4)/4,用到了 3 个 4，是最优方案。

试补全程序。

```
01 #include <iostream>
02 #include <cstdlib>
03 #include <climits>
04
05 using namespace std;
06
07 const int M = 10000;
08 bool Vis[M + 1];
09 int F[M + 1];
10
11 void update(int &x, int y) {
12     if (y < x)
13         x = y;
14 }
15
16 int main() {
17     int n;
18     cin >> n;
19     for (int i = 0; i <= M; i++)
20         F[i] = INT_MAX;
21     ①;
22      int r = 0;
23     while (②) {
24         r++;
25         int x = 0;
26         for (int i = 1; i <= M; i++)
27             if (③)
28                 x = i;
29         Vis[x] = 1;
30         for (int i = 1; i <= M; i++)
31             if (④) {
32                 int t = F[i] + F[x];
33                 if (i + x <= M)
34                     update(F[i + x], t);
35                 if (i != x)
```

```
36                  update(F[abs(i - x)], t);
37              if (i % x == 0)
38                  update(F[i / x], t);
39              if (x % i == 0)
40                  update(F[x / i], t);
41          }
42    }
43    cout << F[n] << endl;
44    return 0;
45 }
```

(1)①处应填（ ）

A. F[4] = 0　　B. F[1] = 4　　C. F[1] = 2　　D.F[4] = 1

(2)②处应填（ ）

A. !Vis[n]　　B. r < n　　C. F[M] == INT_MAX　　D. F[n] == INT_MAX

(3)③处应填（ ）

A. F[i] == r　　B. !Vis[i] && F[i] == r

C. F[i] < F[x]　　D. !Vis[i] && F[i] < F[x]

(4)④处应填（ ）

A. F[i] < F[x]　　B. F[i]<=r　　C. Vis[i]　　D. i <= x

20.（ RMQ 区间最值问题 ）给定序列，a_0，…，a_{n-1}，m 次询问每次询问给定 l,r,求 $\max\{a_l, \ldots, a_r\}$。为了解决该问题，有一个算法叫 the Method of Four Russians，其时间复杂度为 O(n+m)，步骤如下：

- 建立 Cartesian（笛卡尔）树，将问题转化为树上的 LCA（最近公共祖先）问题。
- 对于 LCA 问题，可以考虑其 Euler 序（即按照 DFS 过程，经过所有点，环游回根的序列），即求 Euler 序列上两点间一个新的 RMQ 问题。
- 注意新的问题为 ±1 RMQ，即相邻两点的深度差一定为 1。

下面解决这个 ±1 RMQ 问题，“序列”指 Euler 序列：

- 设 t 为 Euler 序列长度。取 $b=\lceil \log_2 t/2 \rceil$ 将序列每 b 个分为一大块，使用 ST 表（倍增表）处理大块间的 RMQ 问题，复杂度 O(t/b logt)=O(n)。
- （重点）对于一个块内的 RMQ 问题，也需要 O(1) 的算法。由于差分数组 2b−1 种，可以预处理出所有情况下的最值位置，预处理复杂度 O(b2 b)，不超过 O(n)。
- 最终，对于一个查询，可以转化为中间整的大块的 RMQ 问题，以及两端块内的 RMQ 问题。

试补全程序。

```
001 #include <iostream>
002 #include <cmath>
003
004 using namespace std;
005
006 const int MAXN = 100000, MAXT = MAXN << 1;
007 const int MAXL = 18, MAXB = 9, MAXC = MAXT / MAXB;
008
009 struct node {
010    int val;
011    int dep, dfn, end;
012    node *son[2]; // son[0], son[1] 分别表示左右儿子
013 } T[MAXN];
014
015 int n, t, b, c, Log2[MAXC + 1];
016 int Pos[(1 << (MAXB - 1)) + 5], Dif[MAXC + 1];
017 node *root, *A[MAXT], *Min[MAXL][MAXC];
018
019 void build() { // 建立 Cartesian 树
```

```
020     static node *S[MAXN + 1];
021     int top = 0;
022     for (int i = 0; i < n; i++) {
023         node *p = &T[i];
024         while (top && S[top]->val < p->val)
025             ①;
026         if (top)
027             ②;
028         S[++top] = p;
029     }
030     root = S[1];
031 }
032
033 void DFS(node *p) { // 构建 Euler 序列
034     A[p->dfn = t++] = p;
035     for (int i = 0; i < 2; i++)
036         if (p->son[i]) {
037             p->son[i]->dep = p->dep + 1;
038             DFS(p->son[i]);
039             A[t++] = p;
040         }
041     p->end = t - 1;
042 }
043
044 node *min(node *x, node *y) {
045     return ③ ? x : y;
046 }
047
048 void ST_init() {
049     b = (int)(ceil(log2(t) / 2));
050     c = t / b;
051     Log2[1] = 0;
052     for (int i = 2; i <= c; i++)
053         Log2[i] = Log2[i >> 1] + 1;
054     for (int i = 0; i < c; i++) {
055         Min[0][i] = A[i * b];
056         for (int j = 1; j < b; j++)
057             Min[0][i] = min(Min[0][i], A[i * b + j]);
058     }
059     for (int i = 1, l = 2; l <= c; i++, l <<= 1)
060         for (int j = 0; j + l <= c; j++)
061             Min[i][j] = min(Min[i - 1][j], Min[i - 1][j + (l >>
1)]);
062 }
063
064 void small_init() { // 块内预处理
065     for (int i = 0; i <= c; i++)
066         for (int j = 1; j < b && i * b + j < t; j++)
067             if (④)
068                 Dif[i] |= 1 << (j - 1);
069     for (int S = 0; S < (1 << (b - 1)); S++) {
```

```
070         int mx = 0, v = 0;
071         for (int i = 1; i < b; i++) {
072             ⑤;
073            if (v < mx) {
074                 mx = v;
075                 Pos[S] = i;
076             }
077         }
078     }
079 }
080
081 node *ST_query(int l, int r) {
082     int g = Log2[r - l + 1];
083     return min(Min[g][l], Min[g][r - (1 << g) + 1]);
084 }
085
086 node *small_query(int l, int r) { // 块内查询
087      int p = l / b;
088     int S = ⑥;
089     return A[l + Pos[S]];
090 }
091
092 node *query(int l, int r) {
093      if (l > r)
094         return query(r, l);
095      int pl = l / b, pr = r / b;
096     if (pl == pr) {
097         return small_query(l, r);
098     } else {
099         node *s = min(small_query(l, pl * b + b - 1),small_query(pr * b, r));
100          if (pl + 1 <= pr - 1)
101            s = min(s, ST_query(pl + 1, pr - 1));
102         return s;
103     }
104 }
105
106 int main() {
107     int m;
108     cin >> n >> m;
109     for (int i = 0; i < n; i++)
110         cin >> T[i].val;
111     build();
112     DFS(root);
113     ST_init();
114     small_init();
115     while (m--) {
116         int l, r;
117         cin >> l >> r;
118         cout << query(T[l].dfn, T[r].dfn)->val << endl;
119     }
120     return 0;
```

121 }

(1)①处应填（ ）

A. p->son[0] = S[top--]　　B. p->son[1] = S[top--]

C. S[top--]->son[0] = p　　D. S[top--]->son[1] = p

(2)②处应填（ ）

A. p->son[0] = S[top]　　B. p->son[1] = S[top]

C. S[top]->son[0] = p　　D. S[top]->son[1] = p

(3)③处应填（ ）

A. x->dep < y->dep　　B. x < y

C. x->dep > y->dep　　D. x->val < y->val

(4)④处应填（ ）

A. A[i * b + j - 1] == A[i * b + j]->son[0]

B. A[i * b + j]->val < A[i * b + j - 1]->val

C. A[i * b + j] == A[i * b + j - 1]->son[1]

D. A[i * b + j]->dep < A[i * b + j - 1]->dep

(5)⑤处应填（ ）

A. v += (S >> i & 1) ? -1 : 1

B. v += (S >> i & 1) ? 1 : -1

C. v += (S >> (i - 1) & 1) ? 1 : -1

D. v += (S >> (i - 1) & 1) ? -1 : 1

(6)⑥处应填（ ）

A. (Dif[p] >> (r - p * b)) & ((1 << (r - 1)) - 1)

B. Dif[p]

C. (Dif[p] >> (l - p * b)) & ((1 << (r - 1)) - 1)

D. (Dif[p] >> ((p + 1) * b - r)) & ((1 << (r - l + 1)) - 1)