ADS期末辅学题目集——程序填空题答案

程序填空题

5-1 The function `BinQueue_Merge` is to merge two binomial queues `H1` and `H2`, and return `H1` as the resulting queue.

```
BinQueue BinQueue_Merge( BinQueue H1, BinQueue H2 ) {
    BinTree T1, T2, Carry = NULL;
    int i, j;
    H1->CurrentSize += H2-> CurrentSize;
    for ( i=0, j=1; j<= H1->CurrentSize; i++, j*=2 ) {
        T1 = H1->TheTrees[i]; T2 = H2->TheTrees[i];
        switch( 4*!!T1 + 2*!!T2 + !!Carry ) (3分) ) {
        case 0:
        case 4: break;
        case 3:
        case 7:
                Carry = CombineTrees( T2, Carry ) (3分);
                H2->TheTrees[i] = NULL; break;
        case 1:
                H1->TheTrees[i] = Carry; Carry = NULL; break;
        case 2:
                H1->TheTrees[i] = T2; H2->TheTrees[i] = NULL; break;
        case 5:
                Carry = CombineTrees( T1, Carry );
                H1->TheTrees[i] = NULL; break;
        case 6:
                Carry = CombineTrees( T1, T2 );
                H1->TheTrees[i] = H2->TheTrees[i] = NULL; break;
        } /* end switch */
    } /* end for-loop */
    return H1;
}
```

5-2 Suppose we are given n points $p_1, p_2, ... p_n$ located on the x-axis. x_i is the x-coordinate of p_i. Let us further assume that $x_1 = 0$, and the points are given from left to right. These n points determine $\frac{n(n-1)}{2}$ (not-necessarily unique) distances $d_1, d_2, ... d_{n(n-1)/2}$ between every pair of points of the form $|x_i - x_j|$ $(i \neq j)$.

The Turnpike reconstruction problem is to reconstruct a point set from the distances.

This algorithm is to read the number n and $\frac{n(n-1)}{2}$ distances d_i, then print one valid sequence of points p_i. Please complete the following program.

```
#include <algorithm>
#include <cstdio>
const int MAXN = 1000, MAXD = MAXN * (MAXN - 1) / 2;
int p[MAXN], d[MAXD], n, m;
int id[MAXD];
bool used[MAXD];
int binary_search(int x, int m) {
    int l = 0, r = m;
    while (l < r) {
        int mid = (l + r) / 2;
        if (d[mid] < x || (d[mid] == x && used[mid]))
            l = mid + 1 (2分);
        else
            r = mid;
    }
    return l;
}
bool recursive(int now, int top, int m) {
    int i;
    for (i = 0; i < now; i++) {
        id[top + i] = binary_search(abs(p[i] - p[now]), m);
        if ( d[id[top + i]] == abs(p[i] - p[now]) (2分) && !used[id[top + i]])
            used[id[top + i]] = true;
        else break;
    }
    if (i == now) {
        if (now == n - 1)
            return true;
        while (used[m - 1])
            m--;
        p[now + 1] = d[m - 1];
        if (recursive(now + 1, top + now, m))
            return true;
        if (now <= 1)
            return false;
        p[now + 1] =  p[1] - d[m - 1] (2分);
        if (recursive(now + 1, top + now, m))
            return true;
    }
    for(int j = 0; j < i; j++)
        used[id[top + j]] = false (2分);
    return false;
}
int main()
{
    scanf("%d", &n);
    m = n * (n - 1) / 2;
    for (int i = 0; i < m; i++)
        scanf("%d", &d[i]);
    std::sort(d, d + m);
    p[0] = 0;
    if (!recursive( 0, 0, m (2分))) {
        puts("NO ANSWER");
        return 0;
    }
    std::sort(p, p + n);
    for (int i = 0; i < n; i++)
        printf("%d\n", p[i]);
    return 0;
}
```

5-3 An n-digit number that is the sum of the n-th powers of its digits is called an n-narcissistic number. For example, $\mathbf{153}$ is a $\mathbf{3}$-narcissistic number since $\mathbf{153} = \mathbf{1}^3 + \mathbf{5}^3 + \mathbf{3}^3$, and $\mathbf{1634}$ is a $\mathbf{4}$-narcissistic number since $\mathbf{1634} = \mathbf{1}^4 + \mathbf{6}^4 + \mathbf{3}^4 + \mathbf{4}^4$.

Please complete the following program that prints the sum of all n-narcissistic numbers.

```
#include <cstdio>
long long cost[10], ans;
int cnt[10], t[10], n;
void dfs(int rest, int now, long long current) {
    if (rest == 0) {
        long long temp = current;
        for (int i = 0; i < 10; i++)
            t[i] = 0;
        while (temp > 0) {
            ++t[temp % 10];
            temp /= 10;
        }
        bool flag = 1;
        for (int i = 0; i < 10; i++)
            if (cnt[i] != t[i]) {
                flag = 0;
                break;
            }
        if (flag) {
            ans += current (2分);
        }
        return;
    }
    if ( now > 9 (2分)) {
        return;
    }
    for (cnt[now] = 0; cnt[now] <= rest; cnt[now]++)
        dfs(rest - cnt[now], now + 1,  current + cost[now] * cnt[now] (2分));
    cnt[now] (2分) = 0;
}
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < 10; i++) {
        cost[i] = 1;
        for (int j = 0; j < n; j++)
            cost[i] *= i;
    }
    dfs(n, 0, 0 (2分));
    printf("%lld\n",ans);
    return 0;
}
```

5-4 In a permutation A, if there exists a pair of numbers A_i and A_j satisfied $i < j$ and $A_i > A_j$, then A_i and A_j are called an inverted pair.

Giving a permutation of N, please find the number of the inverted pairs in it.

Hint: The function $work(l, r)$ returns the number of inverted pairs in $A_l A_{l+1} \dots A_r$ and makes $A_l A_{l+1} \dots A_r$ being sorted in increasing order.

```
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010;
int tmp[N], a[N];
long long work(int l, int r) {
    if (l == r)
        return 0;
    int mid = (l + r) >> 1;
    long long res = 0;
    res += work(l, mid);
    res += work(mid+1, r) (2分);
    int t1 = l, t2 = mid + 1, tt = l;
    while (t1 <= mid && t2 <= r) (2分){
        if (a[t1] < a[t2])
            tmp[tt++] = a[t1++];
        else {
            res +=  mid - t1 + 1 (3分);
            tmp[tt++] = a[t2++];
        }
    }
    while (t1 <= mid)
        tmp[tt++] = a[t1++];
    while (t2 <= r)
        tmp[tt++] = a[t2++];
    for (int i = l; i <= r; i++)
        a[i] = tmp[i] (2分);
    return res;
}
int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    printf("%lld\n",  work(1, n) (1分));
    return 0;
}
```

5-5 Giving an array of N integers, please calculate the maximum subsegment sum.

```
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 100010;
int a[N];
long long work(int l, int r) {
    if (l == r)
        return  a[l] (2分);
    int mid = (l + r) >> 1;
    long long res = 0;
    res = max(res, work(l, mid));
    res =  max(res, work(mid + 1, r)) (2分);
    long long mxl = 0, suml = 0, mxr = 0, sumr = 0;
    for (int i = mid; i >= l; i--) {
        suml += a[i];
        mxl = max(mxl, suml);
    }
    for (int i = mid + 1; i <= r; i++) {
        sumr += a[i] (2分);
        mxr = max(mxr, sumr) (2分);
    }
    res =  max(res, mxl + mxr) (2分);
    return res;
}
int main()
{
    int T;
    scanf("%d",&T);
    while (T--) {
        int n;
        scanf("%d", &n);
        for(int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        printf("%lld\n", work(1, n));
    }
    return 0;
}
```