

Questions on the algorithm part (Total = 100)

Question 1. [25] A multiset is a set where we allow duplication. For example, $S = \{1, 2, 3, 3\}$ is a multiset. Permutations on a multiset are similar to those on a set, except that identical elements are not distinguishable. The set of permutations, $P(S)$, for the above S is given in lexicographic order as follows:

```
1 2 3 3
1 3 2 3
1 3 3 2
2 1 3 3
2 3 1 3
2 3 3 1
3 1 2 3
3 1 3 2
3 2 1 3
3 2 3 1
3 3 1 2
3 3 2 1
```

Let the multiplicity of i be n_i ($i=1, \dots, k$) and the size of S be n . That is, n is the sum of all n_i . Then the size of $P(S)$ is given by $|P(S)| = n!/(n_1! \dots n_k!)$. In our example, we have $|P(S)| = 4!/(1!1!2!) = 12$.

(1) List up all permutations on the multiset $S = \{1, 2, 2, 3, 3\}$ in lexicographic order. How many permutations are there? [12]

(2) Write a program for generating the set of permutations in lexicographic order for a given multiset S . A program for permutations on a set is given below as a hint. You can assume array $a[1 .. n]$ is initialised to $(1, \dots, 1, 2, \dots, 2, \dots, k, \dots, k)$, where element i appears n_i times for $i=1, \dots, k$. [13]

```
var a: array[1..100] of integer;
procedure swap(i, j);
var w:integer;
begin w:=a[i]; a[i]:=a[j]; a[j]:=w; end;
function minimum(k);
var i,j,temp:integer;
begin
  j:=0; temp:=99;
  for i:=k to n do
    if (a[i]<temp) and (a[i]>a[k-1]) then begin
      temp:=a[i]; j:=i
    end;
  minimum:=j;
end;
procedure reverse(k);
var i:integer;
begin for i:=k to (k+n-1) div 2 do swap(i, n-i+k);end;
```

```

begin {main program}
  readln(n);
  for i:=1 to n do a[i]:=i;
  out;
  repeat
    i:=n;
    while a[i-1]>a[i] do i:=i-1;
    i:=i-1;
    if i>0 then begin
      reverse(i+1);
      j:=minimum(i+1);
      swap(i,j);
      out
    end
  until i=0;
end.

```

Question 2. [25] We consider the problem of generating sequences of well-formed parentheses and brackets. For example, $()()$ and $()[]$ are well-formed, but $([])$ is not. The following is the list of such sequences of two pairs, that is, of length 4. This list is given in lexicographic order, which is defined in the following. The lexicographic order of parenthesis strings is given by $(())$, $()()$. In each string, the first '(' and second '(' are regenerated using '(' and '[' in lexicographic order with '(' < '[' , and the partners are regenerated accordingly.

```

  (( ))
  ([ ])
  [( )]
  [[]]
  ()()
  ()[]
  []()
  [][]

```

(1) List up all such sequences of 3 pairs, that is, of length 6. [13]

(2) Write a program that generate above sequences of n pairs, that is, of length 2n.[12]

Hint. The following program generates parenthesis strings of length 2n, and generates array 'left' and 'right' which give the positions of left parentheses and the corresponding right parentheses. The output for n=4 follows the program.

```

program ex(input,output);
var i,j,k,m,n:integer;
    a,left,right:array[0..100] of integer;
    q:array[0..100] of char;
procedure out;
var i:integer;
begin
    for i:=1 to n do write(a[i]:2);
    for i:=1 to 2*n do write(q[i]:2);
    for i:=1 to n do write(left[i]:2); {position of i-th left parenthesis}
    write(' ');
    for i:=1 to n do write(right[i]:2);{position of the partner of the above}
    readln
end;
procedure produce;
var i,s,t:integer;
    stack:array[0..100] of integer;
begin
    s:=0; t:=0;
    for i:=1 to n do begin
        s:=s+a[i-1];
        q[s+i]:='('; left[i]:=s+i;
        t:=t+1; stack[t]:=i; {push down i into stack}
        for k:=1 to a[i] do begin
            q[s+i+k]:=')'; right[stack[t]]:=s+i+k;
            t:=t-1; {pop up stack} {example (( ( ) ) ), i=4, stack=(1,2,4)}
        end;
        {left=(1,2,3,5), right = (8,7,4,6)}
    end;
end;
out;
procedure paren(i,s:integer);
var k,t:integer;
begin
    if i<=n-1 then begin
        for k:=0 to i-s do begin
            a[i]:=k;
            t:=s+a[i];
            paren(i+1,t);
        end
    end
    else if i=n then begin a[i]:=n-s; produce end
end;
begin
    readln(n);
    paren(0,0)
end.

```

a	q	left	right
0 0 0 4	(((()))	1 2 3 4	8 7 6 5
0 0 1 3	((()))	1 2 3 5	8 7 4 6
0 0 2 2	((()))	1 2 3 6	8 5 4 7
0 0 3 1	((()))	1 2 3 7	6 5 4 8
0 1 0 3	((()))	1 2 4 5	8 3 7 6
0 1 1 2	((()))	1 2 4 6	8 3 5 7
0 1 2 1	((()))	1 2 4 7	6 3 5 8
0 2 0 2	((()))	1 2 5 6	4 3 8 7
0 2 1 1	((()))	1 2 5 7	4 3 6 8
1 0 0 3	(()(()))	1 3 4 5	2 8 7 6
1 0 1 2	(()(()))	1 3 4 6	2 8 5 7
1 0 2 1	(()(()))	1 3 4 7	2 6 5 8
1 1 0 2	(())(()))	1 3 5 6	2 4 8 7
1 1 1 1	(())(()))	1 3 5 7	2 4 6 8

Question 3. [25]

- (1) Draw a picture of the bitonic sorting network that sort the given set of integers in reverse order. [12]
- (2) Trace your network with (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 , 15, 16). [13]

Question 4. [25] We define distance matrix multiplication by $C = AB$, where elements are defined as follows:

$$c_{ij} = \max\{a_{ik} + b_{kj}\}, \quad i=1, \dots, n, j=1, \dots, n.$$

- (1) Draw a graph that has the following distance matrix, where a_{ij} is the cost of edge (i, j) from vertex i to vertex j. Note that -99 is to show non-existence of an edge. [5]

$$A = \begin{matrix} 0 & 2 & 1 & -99 & -99 & -99 \\ -99 & 0 & 6 & 3 & 3 & -99 \\ -99 & -99 & 0 & 4 & 2 & -99 \\ -99 & -99 & -99 & 0 & 5 & 1 \\ -99 & -99 & -99 & -99 & 0 & 2 \\ -99 & -99 & -99 & -99 & -99 & 0 \end{matrix}$$

- (2) Compute A^2 , A^4 and A^8 by repeated squaring. Confirm that the (i, j) element of A^8 gives the longest distance from i to j in this graph. [10]

Note. If the above graph depicts a schedule of some contract work, the maximum value in the resulting matrix in (1) above determines the total time to complete the work. Each edge corresponds to a job and the cost is the time it takes. If there are one or more incoming edges to a vertex, all corresponding jobs must be finished before any job on an outgoing edge from the vertex can start.

- (3) Describe how the above computation can be done in $O(\log^2 n)$ parallel time. [10]

