문제 27 다리(Bridge)

일단 문제 설명과 제한 조건이 단순한 문제이고, 두 명이 건너갔다가 한 명이 돌아오는 것을(사람의 수-2) 만큼 반복해 준 뒤, 나머지 2명이 다시 건너가면 된다는 것까지 생각하기는 쉽다. 하지만 그렇게 하는 경우의 수가 고려해야 하는 사람의 수에 비례하여 기하급수적으로 증가하기 때문에, 모든 경우의 수를 다 계산해서 푸는 방법은 적용이 불가능하다. 적용할 수 있는 휴리스틱 그리디 방법이 여러 가지 있는데, 대부분의 방법은 항상 최적해를 내는 방법은 아니다.

모범 답안에서는 우선 주어진 수들을 오름차순으로 정렬하여, 다리를 건너는데 가장 오래 걸리는 사람을 제일 뒤로 배치한 후, 뒤에서 두 사람씩 차례로 다리를 건너게 하는 방법을 사용한다. 이 두 사람이 다리를 건너게 하기 위하여, 일꾼 개념으로 가장 빨리 건너는 사람 두 사람을 사용하는데, 가장 빠른 사람 두 명의 시간을 a, b, 가장 느린 사람 두명의 시간을 c, d라고 하자. 그러면 다음 두 개의 시나리오를 생각할 수 있다(다른 시나리오는 이 시나리오중 하나에 비해서 무조건 느리다).

1st scenario :
(a,b) 건너기 → a 돌아오기 → (c,d) 건너기 → b 돌아오기 : 시간 = b+a+d+b

2nd scenario :
(a,c) 건너기 → a 돌아오기 → (a,d) 건너기 → a 돌아오기 : 시간 = c+a+d+a

두 시나리오 모두 결과적으로는 가장 느린 사람 두 사람만 다리를 건너게 되고, 가장 빠른 사람 두 사람은 다시 원래 자리로 돌아오게 된다. 따라서, 각 시나리오의 진행 시간 중 짧은 것을 선택하면 되는 것이다.

한번 위 과정을 거치면, 가장 느린 사람이 2명 줄어서 전체 사람 수가 2 줄어들게 된다. 이 과정을 반복하면, 가장 느린 사람이 계속 바뀌면서 사람 수가 계속 줄게 된다.
단, 위 시나리오는 사람이 4명 이상 있어야 가능하므로, 3명 이하일 때는 특별 케이스로 분리하여 수행하여야 한다. (어차피 이 경우에는 최적해가 명확하다)

위의 휴리스틱 알고리즘이 항상 최적해를 낸다는 증명은 조금 어렵다. 증명 말고, 그냥 단순하게 생각해 본다면, '가장 느린 사람은 어차피 다리 너머로 보내야 하며, 이왕 보낼 바에는 느린 사람끼리 보내는 것이 좋고, 그 과정에서 같이 갈 사람은 되도록이면 빠른 사람이 좋다. 따라서 위의 알고리즘은 항상 최적해를 보장한다.' 정도로 할 수 있을 것이다.

```
/* @BEGIN_OF_SOURCE_CODE */

/* @JUDGE_ID: 46288WZ 10037 C "" */

#include <stdio.h>
#include <stdlib.h>

#define MAX_N 1000

int compare( const void *arg1, const void *arg2 )
{
    return (*(int*)arg1) - (*(int*)arg2);
}

void main()
{
  int total_case, i;

  scanf("%d", &total_case);

  for ( i = 0; i < total_case; i++ )
  {
    int n, j;
    int time[MAX_N] = {0};
    int total_time = 0;
    int sequence_size = 0;
    int sequence_num[MAX_N*2-1] = {0};
    int sequence_person[MAX_N*2-1][2] = {0};

    scanf("%d", &n);

    for ( j = 0; j < n; j++ )
      scanf("%d", &time[j]);

    qsort( (void *)time, (size_t)n, sizeof(int), compare );
```

```
while (n > 3)
{
   /* if size is larger than 3...
   we can select between 2 scenarios
   suppose x1, x2 is 1st & 2nd smallest num
   and x3, x4 is 1st & 2nd largest num
   1st scenario : (x1, x2) (x1) (x3, x4) (x2) = x2+x1+x4+x2
   2nd scenario : (x1, x3) (x1) (x1, x4) (x1) = x3+x1+x4+x1
   if (x2+x2) < (x3+x1), we choose 1st scenario, otherwise 2nd. */

   int x1, x2, x3, x4;
   x1 = time[0];
   x2 = time[1];
   x3 = time[n-2];
   x4 = time[n-1];

   if (x2+x2 < x3+x1)
   {
      /* 1st scenario */
      sequence_num[sequence_size] = 2;
      sequence_person[sequence_size][0] = x1;
      sequence_person[sequence_size][1] = x2;
      sequence_size ++;
      sequence_num[sequence_size] = 1;
      sequence_person[sequence_size][0] = x1;
      sequence_size ++;
      sequence_num[sequence_size] = 2;
      sequence_person[sequence_size][0] = x3;
      sequence_person[sequence_size][1] = x4;
      sequence_size ++;
      sequence_num[sequence_size] = 1;
      sequence_person[sequence_size][0] = x2;
      sequence_size ++;
      total_time += x2+x1+x4+x2;
   }
```

```
    else
    {
        /* 2nd scenario */
        sequence_num[sequence_size] = 2;
        sequence_person[sequence_size][0] = x1;
        sequence_person[sequence_size][1] = x3;
        sequence_size ++;
        sequence_num[sequence_size] = 1;
        sequence_person[sequence_size][0] = x1;
        sequence_size ++;
        sequence_num[sequence_size] = 2;
        sequence_person[sequence_size][0] = x1;
        sequence_person[sequence_size][1] = x4;
        sequence_size ++;
        sequence_num[sequence_size] = 1;
        sequence_person[sequence_size][0] = x1;
        sequence_size ++;
        total_time += x3+x1+x4+x1;
    }

    n -= 2;
}

/* n=3 case -> make n=2 */
if ( n == 3 )
{
    sequence_num[sequence_size] = 2;
    sequence_person[sequence_size][0] = time[0];
    sequence_person[sequence_size][1] = time[2];
    sequence_size ++;
    sequence_num[sequence_size] = 1;
    sequence_person[sequence_size][0] = time[0];
    sequence_size ++;
    total_time += time[2]+time[0];

    n--;
```

```
    }

    /* n<=2 : send all and finish */
    sequence_num[sequence_size] = n;
    for ( j = 0; j < n; j++ )
    {
      sequence_person[sequence_size][j] = time[j];
    }
    total_time += time[n-1];
    sequence_size ++;

    /* print all */
    printf("%d₩n", total_time);
    for ( j = 0; j < sequence_size; j++ )
    {
      int k;
      for ( k = 0; k < sequence_num[j]; k++ )
      {
        printf("%d", sequence_person[j][k]);
        if ( k < sequence_num[j] - 1 )
  printf(" ");
        else
  printf("₩n");
      }
    }

    if ( i < total_case - 1 )
      printf("₩n");
  }
}

/* @END_OF_SOURCE_CODE */
```