```
/*
        @(#) skitrip.cpp
        writer: kbokis
        28th PDP Final Fase
*/

#include <cstdio>
#include <cassert>
#include <cstdlib>
#include <climits>
#include <algorithm>

using namespace std;

struct station {
        int min_height, max_height;
};

/*
αποθηκευουμε σε εναν τριγωνικο πινακα τιμες. Ο πίνακας μοιάζει με το γνωστο μας ισορροπημένο
δυαδικό δέντρο.
Στην βαση (line==0), εχουμε τα N στοιχεια εισοδου
Στην επομενη γραμμη (line==1) έχουμε N/2 στοιχεια (αν N αρτιος) ή N/2+1 (αν N περιττός)
Συνεχιζουμε έως ότου να έχουμε μόνο 1 στοιχειο (το οποίο συμβαίνει στην γραμμη last_line).
Κάθε στοιχείο της γραμμης i, με i>0, είναι αποθηκευμένο στη θεση A[line][i] και έχει το
ελάχιστο και μεγιστο στοιχείο απο τα A[line-1][i*2] και αν υπάρχει και του A[line-1][i*2+1].
*/

int N;
const size_t MAXLN = 24;//μεγιστος αριθμός γραμμών που απαιτείται (για 2000000 στοιχεία)
station *A[MAXLN] = {0};//πίνακας πινάκων (δυσδιαστατος δυναμικος πίνακας)
int size[MAXLN] = {0}, last_line=0;//αριθμός στοιχείων της κάθε γραμμής

int findM(int h,int line=last_line,int start=0){
        //αναδρομικη συναρτηση για να βρει το πιο δεξιο στοιχείο που είναι ισο 'η μεγαλύτερο
        απο το h
        //printf("findM(h=%d, line=%d, start=%d, size[line]=%d)\n",h,line,start,size[line]);
        int accepted_i = -1;
        for(int i=start;i<start+2 && i<size[line];i++){
                if(A[line][i].max_height>=h)
                        accepted_i = i;//find rightmost entry in this level that is higher
                        or same level with 'h'
        }
        if(accepted_i>=0 && line)
                return findM(h,line-1,accepted_i*2);
        return accepted_i;
}

int main(){
        FILE *in = fopen("skitrip.in","r"), *out = fopen("skitrip.out","w");
        assert(in && out);
        fscanf(in,"%d",&N);
        assert(N>0);

        {//allocate tree heap space
                int n = N;//elements for current line/level
                for(last_line=0;last_line<MAXLN && n>0;last_line++){
```

```
                        size[last_line] = n;
                        A[last_line] = new station[n];
                        if(n>2 && n%2)
                                n++;
                        n/=2;
                }
                last_line--;
        }

        for(int i=0;i<N;i++){
                fscanf(in,"%d",&A[0][i].min_height);
                A[0][i].max_height = A[0][i].min_height;
        }

        //dp:fill the binary tree
        for(int line=1;size[line];line++){
                //we have to calc minimums on N items in size[line] items
                for(int i=0;i<size[line];i++){
                        A[line][i].min_height = A[line-1][i*2].min_height;
                        A[line][i].max_height = A[line-1][i*2].max_height;
                        if(i*2+1<size[line-1]){
                                A[line][i].min_height =
                                min(A[line][i].min_height,A[line-1][i*2+1].min_height);
                                A[line][i].max_height =
                                max(A[line][i].max_height,A[line-1][i*2+1].max_height);
                        }
                }
        }

        #if !defined(CONTEST) && 0
        //εκτυπωση στην οθονη για έλεγχο του δένδρου
        for(int line=0;size[line];line++){
                printf("Line %2d, Size=%2d: ",line,size[line]);
                for(int i=0;i<size[line];i++){
                        printf("[%3d,%3d] ",A[line][i].min_height,A[line][i].max_height);
                }
                printf("\n");
        }
        #endif

        int maxdist = 0;
        for(int i=0;i<N;i++){
                int M = findM(A[0][i].min_height,last_line,0);
                if(M>0 && M - i > maxdist)
                        maxdist = M - i;
        }
        #ifndef CONTEST
                printf("%d\n",maxdist);
        #endif
        fprintf(out,"%d\n",maxdist);

        return 0;
}
```