#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

int parse\_int(char\*);

int cmpfunc(const void\* a,const void\* b){

    long long int x=\*(long long int\*)a;

    long long int y=\*(long long int\*)b;

    if(x<y) return -1;

    else if(x>y) return 1;

    return 0;

}

/\*

 \* Complete the 'maxMin' function below.

 \*

 \* The function is expected to return an INTEGER.

 \* The function accepts following parameters:

 \*  1. INTEGER k

 \*  2. INTEGER\_ARRAY arr

 \*/

int maxMin(int k, int n,long long  int\* arr) {

qsort(arr,n,sizeof(long long int ),cmpfunc);

long long int unfairness=LLONG\_MAX;

for(int i=0;i<=n-k;i++){

    long long int diff=arr[i+k-1]-arr[i];

    if(diff<unfairness) unfairness=diff;

}

return (int)unfairness;

}

int main()

{

    FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

    int n = parse\_int(ltrim(rtrim(readline())));

    int k = parse\_int(ltrim(rtrim(readline())));

    int\* arr = malloc(n \* sizeof(int));

    for (int i = 0; i < n; i++) {

        int arr\_item = parse\_int(ltrim(rtrim(readline())));

        \*(arr + i) = arr\_item;

    }

    int result = maxMin(k, n, arr);

    fprintf(fptr, "%d\n", result);

    fclose(fptr);

    return 0;

}

char\* readline() {

    size\_t alloc\_length = 1024;

    size\_t data\_length = 0;

    char\* data = malloc(alloc\_length);

    while (true) {

        char\* cursor = data + data\_length;

        char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

        if (!line) {

            break;

        }

        data\_length += strlen(cursor);

        if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') {

            break;

        }

        alloc\_length <<= 1;

        data = realloc(data, alloc\_length);

        if (!data) {

            data = '\0';

            break;

        }

    }

    if (data[data\_length - 1] == '\n') {

        data[data\_length - 1] = '\0';

        data = realloc(data, data\_length);

        if (!data) {

            data = '\0';

        }

    } else {

        data = realloc(data, data\_length + 1);

        if (!data) {

            data = '\0';

        } else {

            data[data\_length] = '\0';

        }

    }

    return data;

}

char\* ltrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    while (\*str != '\0' && isspace(\*str)) {

        str++;

    }

    return str;

}

char\* rtrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    char\* end = str + strlen(str) - 1;

    while (end >= str && isspace(\*end)) {

        end--;

    }

    \*(end + 1) = '\0';

    return str;

}

int parse\_int(char\* str) {

    char\* endptr;

    int value = strtol(str, &endptr, 10);

    if (endptr == str || \*endptr != '\0') {

        exit(EXIT\_FAILURE);

    }

    return value;

}