Aluno: Jonathan (12/0122553)

Aluno: Laércio (13/0119105)

Disciplina: Estrutura de Dados e Algoritmos 2

Professor: Maurício Serrano

Data: 01/05/2017

Lista 1

1. Five pirates have obtained 100 gold coins and have to divide up the loot. The pirates are all extremely intelligent, treacherous and selfish (especially the captain). The captain always proposes a distribution of the loot. All pirates vote on the proposal, and if half the crew or more go "Aye", the loot is divided as proposed, as no pirate would be willing to take on the captain without superior force on their side. If the captain fails to obtain support of at least half his crew (which includes himself), he faces a mutiny, and all pirates will turn against him and make him walk the plank. The pirates start over again with the next senior pirate as captain.
   What is the maximum number of coins the captain can keep without risking his life?

   - **R**: São 5 piratas, e a regra é que 50% ou mais do clã precisam aceitar a proposta de forma que o capitão não seja jogado ao mar. Caso a maioria do clã não aceite, o próximo capitão será o pirata mais velho. O objetivo é ter o maior número de moedas para o capitão sem que o mesmo arrisque sua vida.

     (a) Enumerando 5 piratas, do mais velho para o mais novo:
        - `P = p1, p2, p3, p4, p5`

     (b) Assumindo a gânancia máxima, a divisão seria:
        - `D = 100, 0, 0, 0, 0`
        - Os 4 restantes votariam contra a decisão, e o capitão seria jogado ao mar.

     (c) `p2` sempre votará contra pois o mesmo será o próximo capitão e poderá decidir a divisão, não faz sentido dividir algo com ele.

     (d) `p3` votará a favor com muita facilidade, pois se `p2` for eleito o capitão, ele acabará ficando sem nada da distribuição (de acordo com a afirmação anterior).

(e) `p4` sabe que se `p2` for capitão, ele certamente ficará com uma fatia, visto que ele não oferece riscos à liderança de `p2`, e com apenas ele já cumpriria a meta dos 50%.

(f) `p5` votará a favor com muita facilidade, pois se `p2` for eleito o capitão, o total de membros se tornará 4, e muito provavelmente `p4` receberá a única fatia de divisão, suficiente para cumprir a meta dos 50%.

(g) Com isso, assumindo a ganância máxima com segurança, a divisão seria:

   – `D = 98, 0, 1, 0, 1`

   – `p2` sempre votará contra

   – `p3` sabe que 1 é melhor que nada

   – `p4` sabe que conseguirá maior ou igual valor votando contra

   – `p5` sabe que 1 é melhor que nada

2. The five pirates mentioned previously are joined by a sixth, then plunder a ship with only one gold coin. After venting some of their frustration by killing all on board the ship, they now need to divvy up the one coin. They are so angry, they now value in priority order: 1) Their lives; 2) Getting money; 3) Seeing other pirates die. So if given the choice between two outcomes, in which they get the same amount of money, they'd choose the outcome where they get to see more of the other pirates die. How can the captain save his skin?

   • **R**: Agora são 6 piratas e um recurso mínimo de 1 moeda.

   (a) Enumerando 6 piratas, do mais velho para o mais novo:
      – `P = p1, p2, p3, p4, p5, p6`

   (b) As possíveis divisões seriam:
      – `D = 1, 0, 0, 0, 0, 0`
         ∗ Caso este que certamente levaria o capitão ao mar

      – `D = 0, 1, 0, 0, 0, 0`
         ∗ Cumprirá apenas 2 votos de um total de 3 necessários

      – `D = 0, 0, 1, 0, 0, 0`

* p2 sabe que se p1 for jogado ao mar, ele certamente será jogado, pois nunca conseguirá 3 votos em um grupo de 5 com apenas 1 moeda. Ele certamente votará a favor para salvar sua pele

* p3 sabe que se p1 e p2 for jogado ao mar, ele sobreviverá mas certamente perderá a moeda, caso este que agora possui

* O capitão consegue os votos necessários nessa condição e salvará sua pele

3. Implemente o Merge Sort de forma recursiva, ou seja, que divide o vetor aproximadamente ao meio em cada passo e que encerra a recursão quando obtém um vetor de apenas uma posição (que, por definição, já está ordenado). Apresente de forma gráfica as trocas enquanto elas ocorrem.

   - **R**:
     - Código completo no arquivo `q3_merge.cpp`
     - Compilar o código via makefile: `make q3`
     - Necessário GNUplot (`sudo apt-get install gnuplot` - Ubuntu)

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <vector>
#include "io.h"
#include "list_creator.h"
#include "gnuplot_i.h"

using namespace std;

void merge_sort(vector<int> &elements, vector<int>::iterator start,
    vector<int>::iterator finish);
void merge(vector<int> &elements, vector<int>::iterator start,
    vector<int>::iterator mid, vector<int>::iterator finish);

gnuplot_ctrl *plot;

int main(int argc, char* argv[])
{
    // Check Arguments Quantity
    if(argc != 3)
    {
        inform_execution_pattern();
        inform_quantity_of_elements_pattern();
        inform_seed_pattern();
        exit(EXIT_FAILURE);
    }

    // Create List
    int quantity_of_elements = atoi(argv[1]);
    int seed = atoi(argv[2]);
    vector<int> elements;
    if(!create_list(elements, quantity_of_elements, seed))
    {
        exit(EXIT_FAILURE);
    }

    print_list("Initial List", elements);
    print_rule();

    plot = gnuplot_init();
    merge_sort(elements, begin(elements), end(elements));
    sleep(3);
    gnuplot_close(plot);
    print_list("Sorted List", elements);

    return 0;
}

void merge_sort(vector<int> &elements, vector<int>::iterator start,
    vector<int>::iterator finish)
{
    size_t length = distance(start, finish);
    if(length > 1)
    {
        size_t middle = length/2;
        vector<int>::iterator mid = next(start, middle);
        merge_sort(elements, start, mid);
        merge_sort(elements, mid, finish);
        merge(elements, start, mid, finish);
    }
}

void merge(vector<int> &elements, vector<int>::iterator start,
    vector<int>::iterator mid, vector<int>::iterator finish)
{
    gnuplot_resetplot(plot);
    gnuplot_setstyle(plot, "impulses");
```

```
char command[100];
sprintf(command, "set xrange [-1:%lu]", distance(start, finish));
gnuplot_cmd(plot, command);
double points[distance(start, finish)];
copy(start, finish, points);
char description[100];
print_step(description, "STEP", elements, start, finish);
gnuplot_plot_x(plot, points, distance(start, finish), description);
sleep(2);

vector<int> work_vector(start,finish);

vector<int>::iterator work_start = begin(work_vector);
vector<int>::iterator work_finish = end(work_vector);
vector<int>::iterator work_mid = next(work_start, distance(start, mid));

vector<int>::iterator left = work_start;
vector<int>::iterator right = work_mid;
vector<int>::iterator index = start;

while(left < work_mid && right < work_finish)
{
    if(*left < *right)
    {
        *index = move(*left++);
    }
    else
    {
        *index = move(*right++);
    }
    ++index;
}

while(left < work_mid)
{
    *index = move(*left++);
    ++index;
}

while(right < work_finish)
{
    *index = move(*right++);
    ++index;
}

gnuplot_resetplot(plot);
```

```
    gnuplot_setstyle(plot, "impulses");
    sprintf(command, "set xrange [-1:%lu]", distance(start, finish));
    gnuplot_cmd(plot, command);
    copy(start, finish, points);
    print_step(description, "SORT", elements, start, finish);
    gnuplot_plot_x(plot, points, distance(start, finish), description);
    sleep(2);
    print_rule();
}
```

4. Implemente o Radix Sort utilizando o Counting Sort. Comece pelo algarismo mais significativo, utilizando os baldes do Bucket Sort para garantir o funcionamento correto.

   - **R**:
     - Código completo no arquivo `q4_radix.cpp`
     - Compilar o código via makefile: `make q4`

```
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include "io.h"
#include "list_creator.h"

using namespace std;

void radix_sort(vector<int> &elements);
void counting_sort(vector<vector<int>> &buckets, int digit);

int main(int argc, char* argv[])
{
    // Check Arguments Quantity
    if(argc != 3)
    {
        inform_execution_pattern();
        inform_quantity_of_elements_pattern();
        inform_seed_pattern();
        exit(EXIT_FAILURE);
    }

    // Create List
    int quantity_of_elements = atoi(argv[1]);
    int seed = atoi(argv[2]);
    vector<int> elements;
    if(!create_list(elements, quantity_of_elements, seed))
    {
        exit(EXIT_FAILURE);
    }

    print_list("Initial List", elements);

    radix_sort(elements);

    print_list("Sorted List", elements);

    return 0;
}

void radix_sort(vector<int> &elements)
{
    vector<vector<int>> buckets(100);

    for(int element : elements)
    {
        buckets[element/10].push_back(element);
    }

    for(int digit = 3; digit > 0; --digit)
    {
        counting_sort(buckets, digit);
    }

    int index = 0;
    for(vector<int> bucket : buckets)
    {
        for(int element : bucket)
        {
            elements[index] = element;
            index++;
        }
    }
}

void counting_sort(vector<vector<int>> &buckets, int digit)
{
    unsigned int fraction = 1;
    unsigned int module = 10;
    for(int i = 0; i < digit-1; i++)
    {
        fraction *= 10;
        module *= 10;
    }

    for(vector<int> bucket : buckets)
    {
        if(bucket.size() > 0)
        {
            vector<int> histogram(10,0);
            vector<int> sorted_bucket(bucket.size());
            for(unsigned int j = 0; j < bucket.size(); ++j)
            {
                unsigned int index = (bucket[j]%module)/fraction;
                ++histogram[index];
            }
            for(unsigned int i = 1; i < 10; ++i)
            {
                histogram[i] = histogram[i] + histogram[i-1];
            }
            for(int j = bucket.size(); j > 0; --j)
            {
                int histogram_index = (bucket[j-1]%module)/fraction;
                int index = histogram[histogram_index]-1;
                sorted_bucket[index] = bucket[j-1];
                --histogram[histogram_index];
            }
            bucket = sorted_bucket;
        }
    }
}
```

5. Gere o código de Huffman para o nome de um jogo e uma série de TV de sua preferência. Os nomes devem conter pelo menos dez letras.

   - **R**:
     - Passos para FINAL FANTASY (Jogo): `q5_huffman_game.txt`
     - Passos para JESSICA JONES (Série): `q5_huffman_series.txt`

```
=============
FINAL FANTASY
=============
LENGTH: 13
p(F).2
p(I).1
p(N).2
p(A).3
p(L).1
```

```
 p( ).1
 p(T).1
 p(S).1
 p(Y).1

--> STEP 0
 p(I).1
 p(L).1
 p( ).1
 p(T).1
 p(S).1
 p(Y).1
 p(F).2
 p(N).2
 p(A).3

--> STEP 1
 p( ).1
 p(T).1
 p(S).1
 p(Y).1
 .2
 |-- p(I).1
 `-- p(L).1
 p(F).2
 p(N).2
 p(A).3

--> STEP 2
 p(S).1
 p(Y).1
 .2
 |-- p( ).1
 `-- p(T).1
 .2
 |-- p(I).1
 `-- p(L).1
 p(F).2
 p(N).2
 p(A).3

--> STEP 3
 .2
 |-- p(S).1
 `-- p(Y).1
 .2
 |-- p( ).1
 `-- p(T).1
 .2
 |-- p(I).1
 `-- p(L).1
 p(F).2
 p(N).2
 p(A).3

--> STEP 4
 .2
 |-- p(I).1
 `-- p(L).1
 p(F).2
 p(N).2
 p(A).3
 .4
 |-- .2
 |   |-- p(S).1
 |   `-- p(Y).1
 `-- .2
     |-- p( ).1
     `-- p(T).1

--> STEP 5
 p(N).2
 p(A).3
 .4
 |-- p(F).2
 `-- .2
     |-- p(I).1
     `-- p(L).1
 .4
 |-- .2
 |   |-- p(S).1
 |   `-- p(Y).1
 `-- .2
     |-- p( ).1
     `-- p(T).1

--> STEP 6
 .4
 |-- p(F).2
 `-- .2
     |-- p(I).1
     `-- p(L).1
 .4
 |-- .2
 |   |-- p(S).1
 |   `-- p(Y).1
 `-- .2
     |-- p( ).1
     `-- p(T).1
 .5
 |-- p(N).2
 `-- p(A).3

--> STEP 7
 .5
 |-- p(N).2
 `-- p(A).3
 .8
 |-- .4
 |   |-- p(F).2
 |   `-- .2
 |       |-- p(I).1
 |       `-- p(L).1
 `-- .4
     |-- .2
     |   |-- p(S).1
     |   `-- p(Y).1
     `-- .2
         |-- p( ).1
         `-- p(T).1

--> STEP 8
 .13
 |-- .5
 |   |-- p(N).2
 |   `-- p(A).3
 `-- .8
     |-- .4
     |   |-- p(F).2
     |   `-- .2
     |       |-- p(I).1
     |       `-- p(L).1
     `-- .4
         |-- .2
         |   |-- p(S).1
         |   `-- p(Y).1
         `-- .2
             |-- p( ).1
```

```
            `-- p(T).1

--> FINAL
.13
1-- .5
|   1-- p(N).2
|   0-- p(A).3
0-- .8
    1-- .4
    |   1-- p(F).2
    |   0-- .2
    |       1-- p(I).1
    |       0-- p(L).1
    0-- .4
        1-- .2
        |   1-- p(S).1
        |   0-- p(Y).1
        0-- .2
            1-- p( ).1
            0-- p(T).1
F = 011
I = 0101
N = 11
A = 10
L = 0100
_ = 0001
T = 0000
S = 0011
Y = 0010

RESULT: 0110101111001000001011101100001000110010

=============
JESSICA JONES
=============
LENGTH: 13
p(J).2
p(E).2
p(S).3
p(I).1
p(C).1
p(A).1
p( ).1
p(O).1
p(N).1
```

```
--> STEP 0
 p(I).1
 p(C).1
 p(A).1
 p( ).1
 p(O).1
 p(N).1
 p(J).2
 p(E).2
 p(S).3

--> STEP 1
 p(A).1
 p( ).1
 p(O).1
 p(N).1
 .2
 |-- p(I).1
 `-- p(C).1
 p(J).2
 p(E).2
 p(S).3

--> STEP 2
 p(O).1
 p(N).1
 .2
 |-- p(A).1
 `-- p( ).1
 .2
 |-- p(I).1
 `-- p(C).1
 p(J).2
 p(E).2
 p(S).3

--> STEP 3
 .2
 |-- p(O).1
 `-- p(N).1
 .2
 |-- p(A).1
 `-- p( ).1
 .2
 |-- p(I).1
 `-- p(C).1
 p(J).2
 p(E).2
 p(S).3

--> STEP 4
 .2
 |-- p(I).1
 `-- p(C).1
 p(J).2
 p(E).2
 p(S).3
 .4
 |-- .2
 |   |-- p(O).1
 |   `-- p(N).1
 `-- .2
     |-- p(A).1
     `-- p( ).1

--> STEP 5
 p(E).2
 p(S).3
 .4
 |-- p(J).2
 `-- .2
     |-- p(I).1
     `-- p(C).1
 .4
 |-- .2
 |   |-- p(O).1
 |   `-- p(N).1
 `-- .2
     |-- p(A).1
     `-- p( ).1

--> STEP 6
 .4
 |-- p(J).2
 `-- .2
     |-- p(I).1
     `-- p(C).1
 .4
 |-- .2
 |   |-- p(O).1
 |   `-- p(N).1
 `-- .2
```

```
    |-- p(A).1
    `-- p( ).1
.5
|-- p(E).2
`-- p(S).3

--> STEP 7
.5
|-- p(E).2
`-- p(S).3
.8
|-- .4
|   |-- p(J).2
|   `-- .2
|       |-- p(I).1
|       `-- p(C).1
`-- .4
    |-- .2
    |   |-- p(O).1
    |   `-- p(N).1
    `-- .2
        |-- p(A).1
        `-- p( ).1

--> STEP 8
.13
|-- .5
|   |-- p(E).2
|   `-- p(S).3
`-- .8
    |-- .4
    |   |-- p(J).2
    |   `-- .2
    |       |-- p(I).1
    |       `-- p(C).1
    `-- .4
        |-- .2
        |   |-- p(O).1
        |   `-- p(N).1
        `-- .2
            |-- p(A).1
            `-- p( ).1

--> FINAL
.13
0-- .5
```

```
|   0-- p(E).2
|   1-- p(S).3
1-- .8
    0-- .4
    |   0-- p(J).2
    |   1-- .2
    |       0-- p(I).1
    |       1-- p(C).1
    1-- .4
        0-- .2
        |   0-- p(O).1
        |   1-- p(N).1
        1-- .2
            0-- p(A).1
            1-- p( ).1
J = 100
E = 00
S = 01
I = 1010
C = 1011
A = 1110
_ = 1111
O = 1100
N = 1101

RESULT: 1000001011010101111101111100110011010001
```

6. Altere o código da mediana em tempo linear para utilizar grupos de sete elementos ao invés de cinco.

   - **R**:
     - Código completo no arquivo `q6_median.cpp`
     - Compilar o código via makefile: `make q6`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

#define TRUE 1
#define FALSE 0

const int SIZE_NUMBERS = 10003;

// sorts a small group with insertion sort
```

```c
// group has 7 itens or less
void sort_group(int *numbers, int i, int group_size) {

    for (int j = i + 1; j < i + group_size; j++) {
        int k = j;
        while ((k > i) && (numbers[k] < numbers[k-1])) {
            int aux = numbers[k-1];
            numbers[k-1] = numbers[k];
            numbers[k] = aux;
            k--;
        }
    }
}

// breaks numbers into groups of seven itens
// last group may be not complete
void divide_groups(int *numbers, int size) {

    int i = 0;

    /*  if size = 20
     *  two groups of 7 itens
     *  last group has 6 itens
     */

    for (i = 0; i <= (size - 7); i = i + 7) {
        sort_group(numbers, i, 7);
    }

    int last_size = size % 7; // last group size

    if (last_size > 0) {
        sort_group(numbers, i, last_size);
    }
    else {
        // do nothing, there is no incomplete group
    }

}

// prints numbers in groups of seven itens
// each line is a group
// last line may be incomplete
void print_numbers(int *numbers, int size) {

    for (int i = 0; i < size; i = i + 7) {
        for (int j = 0; (j < 7) && (i + j < size); j++) {
            printf("%d ", numbers[i + j]);
        }
        printf("\n");
    }

    printf("\n");
}

// initiates a array with random numbers
// guarantees only different numbers
void randomize_input(int *input_numbers, int size){

    // numbers are generated from 0 to MAX_NUMBER-1
    const int MAX_NUMBER = SIZE_NUMBERS * 7;

    // array to mark already generated numbers
    int generated[MAX_NUMBER];

    // marks all numbers as "not generated"
    for (int i = 0; i < MAX_NUMBER; i++) {
        generated[i] = FALSE;
    }

    // generates a different number for each position
    for (int i = 0; i < size; i++) {

        int is_different = FALSE;
        int rand_number = 0;

        // wait for a different number
        do {
            rand_number = rand() % MAX_NUMBER;

            if (generated[rand_number] == FALSE) {
                is_different = TRUE;
                generated[rand_number] = TRUE;
            }
        } while (!is_different);

        // finally initiates position with different number
        input_numbers[i] = rand_number;
    }
}

// copy median of each group to a second array
```

```c
// caution with last group
void medians_copy(int *source, int source_size,
    int *destiny, int destiny_size) {

    int number_of_complete_groups = source_size / 7;
    int i = 0;

    // copy medians of complete groups
    for (i = 0; i < number_of_complete_groups; i++) {
        destiny[i] = source[i * 7 + 3];
    }

    // from now on, deals with last group

    int last_group_size = source_size % 7;

    if (last_group_size > 0) {

        // last group first position
        int first_position = i * 7;

        int last_median = 0;

        // defines position of last median accordingly to group size
        // extract its value
        switch (last_group_size) {
            case 1:
            case 2:
                last_median = source[first_position];
                break;
            case 3:
            case 4:
                last_median = source[first_position + 1];
                break;
            case 5:
            case 6:
                last_median = source[first_position + 2];
                break;
            default: // last group is not bigger than 6 itens
                assert(FALSE);
                break;
        }

        // copy last median to second array
        destiny[destiny_size - 1] = last_median;
    }
    else {
        // do nothing, has only complete groups
    }
}

// copies left or right partition to a new array
int* partition_copy(int *source, int start, int end) {

    int destiny_size = (end + 1) - start;
    int *destiny = (int*) malloc(sizeof(int)*(destiny_size));

    for (int i = start; i <= end; i++) {
        destiny[i - start] = source[i];
    }

    return destiny;
}

// simple swap of two numbers of array
void swap(int *input, int first_index, int second_index) {
    int aux = input[first_index];
    input[first_index] = input[second_index];
    input[second_index] = aux;
}

// finds first occurence of value
int find_index(int *input, int size, int value) {

    int value_index = -1;

    for (int i = 0; i < size; i++) {
        if (input[i] == value) {
            value_index = i;
            break;
        }
    }

    return value_index;
}

// smallers numbers to the left, bigger numbers to the right
// [small numbers][mom][bigger numbers]
// has problems with very small input, 1 or 2 itens
void partition_numbers_around_mom (int *input_numbers,
    int input_numbers_size, int median_of_medians) {
```

```
    // short-circuit, fix me!
    if (input_numbers_size == 2) {
        if (input_numbers[0] > input_numbers[1]) {
            swap(input_numbers, 0, 1);
        }
        return;
    }
    else if (input_numbers_size == 1) {
        return;
    }

    int mom_index = find_index(input_numbers,
        input_numbers_size, median_of_medians);
    swap(input_numbers, 0, mom_index);

    int i = 1;
    int j = input_numbers_size - 1;

    while (i < j) {

        // finds a greater in the left side
        while (input_numbers[i] < median_of_medians) {
            i++;
        }

        // finds a smaller in the right side
        while (input_numbers[j] > median_of_medians) {
            j--;
        }

        swap(input_numbers, i, j);
    }

    // both numbers on the middle are inverted
    // mom is at index 0
    swap(input_numbers, 0, i-1);
    swap(input_numbers, 0, i);

    print_numbers(input_numbers, input_numbers_size);
}

// oracle implements Median of Medians algorithm
int ask_oracle_for_median(int *input_numbers, int input_numbers_size) {

    divide_groups(input_numbers, input_numbers_size);
    print_numbers(input_numbers, input_numbers_size);
```

```c
int last_medians_size = input_numbers_size / 7;

if ((input_numbers_size % 7) > 0) {
    last_medians_size = last_medians_size + 1;
}
else {
    // do nothing
}

int *last_medians = (int*) malloc(sizeof(int)*(last_medians_size));
medians_copy(input_numbers, input_numbers_size,
    last_medians, last_medians_size);

while (last_medians_size > 1) {

    printf("LastMedian\n");
    print_numbers(last_medians, last_medians_size);

    divide_groups(last_medians, last_medians_size);

    printf("Sorted LastMedian\n");
    print_numbers(last_medians, last_medians_size);

    int current_medians_size = 0;

    if ((last_medians_size % 7) > 0) {
        current_medians_size = (last_medians_size / 7) + 1;
    }
    else {
        current_medians_size = last_medians_size / 7;
    }

    int *current_medians =
            (int*) malloc(sizeof(int)*(current_medians_size));

    medians_copy(last_medians, last_medians_size,
        current_medians, current_medians_size);

    free(last_medians);

    last_medians = current_medians;
    last_medians_size = current_medians_size;
}

return last_medians[0]; // this is the Median of Medians
```

```c
}

/* asks the oracle for a good candidate to be the median
   partition numbers around median
   checks if candidate is real median
   if not, select left or right partition
 */
int exoteric_select (int *input_numbers, int input_numbers_size,
    int k_esimo) {

    printf("Exoteric Select\n");
    printf("K_esimo: %d\n", k_esimo);

    int mom = ask_oracle_for_median(input_numbers, input_numbers_size);

    partition_numbers_around_mom(input_numbers, input_numbers_size, mom);
    int mom_index = find_index(input_numbers, input_numbers_size, mom);

    printf("Median: %d\n", mom);
    printf("Median index: %d\n\n", mom_index);

    int expected_size = k_esimo;
    int median = -1;

    if (mom_index > expected_size) {

        // oracle guessed too high

        printf("Recursive on L\n");

        int left_size = mom_index;

        int *left_partition = partition_copy(input_numbers, 0, left_size - 1);
        printf("Left partition:\n");
        print_numbers(left_partition, left_size);

        median = exoteric_select(left_partition, left_size, k_esimo);
    }
    else if (mom_index < expected_size) {

        // oracle guessed too low

        printf("Recursive on R\n");

        int right_size = (input_numbers_size - 1) - mom_index;
        int partition_start = mom_index + 1;
        int *right_partition = partition_copy(input_numbers,
            partition_start, input_numbers_size - 1);
        printf("Right partition:\n");
        print_numbers(right_partition, right_size);

        k_esimo = k_esimo - (mom_index + 1);
        median = exoteric_select(right_partition,
            (input_numbers_size - 1) - mom_index, k_esimo);
    }
    else {

        // oracle is correct!

        printf("Median is %d.\n", mom);
        printf("Index is %d.\n", mom_index);
        median = mom;
    }

    return median;
}

// runs 200 times
// compares exoteric_select/MOM with the median of sorted input
int main(int argc, char *argv[]) {

    int times_correct = 0;

    for (int i = 0; i < 200; i++) {

        srand(time(NULL));

        int input_numbers_size = SIZE_NUMBERS;
        assert(input_numbers_size > 0);

        int *input_numbers = (int*) malloc(sizeof(int)*(input_numbers_size));

        randomize_input(input_numbers, input_numbers_size);

        // creates copy of input
        // finds median cheating - sorts the input, O(n^2)
        int *aux = partition_copy(input_numbers, 0, input_numbers_size - 1);
        sort_group(aux, 0, input_numbers_size);
        // sort_group also works with large groups

        printf("Input\n");
        printf("Size: %d\n", input_numbers_size);
        print_numbers(input_numbers, input_numbers_size);

        int k_esimo = (input_numbers_size - 1) / 2; // median is at middle

        // finds median in linear time, O(C.n)
        // C is a big constant, but still linear
        // uses exoteric select and median-of-medians algorithms
        int median = exoteric_select(input_numbers,
            input_numbers_size, k_esimo);

        print_numbers(aux, input_numbers_size);
        printf("Expected Median: %d\n", aux[k_esimo]);
        printf("Median: %d\n", median);

        // compares O(C.n) and cheated O(n^2) solution
        if (median == aux[k_esimo]) {

            // annotates one more right solution
            times_correct++;
        }

        free(input_numbers);
        free(aux);
    }

    printf("Times correct: %d\n", times_correct);

    return 0;
}
```