Procure na internet mecanismos que possibilitem medir tempos de execução de rotinas computacionais. Geralmente, estas medidas são realizadas com o auxílio de funções em C que lêem a hora no sistema (sistemas Unix e Windows geralmente usam funções diferentes). Utilizando os conhecimentos que você obteve com sua pesquisa, meça os tempos de execução das implementações que você criou para os dois problemas de ordenação anteriores e compare os resultados obtidos.

Para fazer uso da função que irá retornar o tempo de execução de um programa é necessário chamar a biblioteca "time.h". Para chamar essa biblioteca basta pôr no cabeçalho do programa:

```
#include <time.h>
```

A partir dessa biblioteca, é possível fazer uso da função "clock", das variáveis do tipo "clock_t", e da macro "CLOCKS_PER_SEC".

A função "clock" retorna o tempo de execução exato do momento em que ela foi chamada. Para encontrar o tempo de execução de um programa precisamos usar ela duas vezes, uma para capturar o tempo inicial e outra para capturar o tempo final da execução.

Se fizermos o tempo final - tempo inicial teremos o tempo de execução do programa em milissegundos. Dividindo esse valor pelo "CLOCKS_PER_SEC" teremos este valor em segundos, pois essa constante tem o valor de 1000000. Para obter o valor em milissegundos, pode-se dividir o "CLOCKS_PER_SEC" por 1000. Nesse contexto, a variável que irá armazenar o valor do tempo da função "clock" deve ser do tipo "clock_t".

Programa de ordenação da Questão 13 com implementação dos recursos da biblioteca time.h para medir o tempo de execução da função de ordenação feita pelo grupo:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h> //clock(), CLOCKS_PER_SEC e clock_t
void crescente (int n){
  float *x;
  int i, j;
  float aux;
  x = malloc(n*sizeof(float));
  for(i=0; i<n; i++){
    printf("Escreva o número %d ", i+1);
    printf("\n");
    scanf("%f", &x[i]);
  }
  for(i=0; i<n; i++){
    for(j=i+1; j<n; j++){
      if(x[i] > x[j]){
        aux = x[i];
        x[i] = x[j];
        x[j] = aux;
      }
    }
  }
  for(i=0; i<n; i++){
    printf("%f, ", x[i]);
  }
  free(x);
}
int main(void) {
  clock_t t; //variável para armazenar tempo
  int n;
  printf("Digite a quantidade de números que deseja ordenar: \n");
  scanf("%d", &n);
  //Verificando tempo de execução da funçaõ de ordenação
  t = clock(); //Armazena o tempo
  crescente(n);
   t = clock() - t; //tempo final - tempo inicial
   printf("Tempo de Execução = %f ms",
((double)t)/((CLOCKS_PER_SEC/1000))); //conversão para double
  return 0;
}
```

Output:

```
clang-7 -pthread -lm -o main main.c
./main
Digite a quantidade de números que deseja ordenar:
4
Escreva o número 1
3.14
Escreva o número 2
2.67
Escreva o número 3
1.602
Escreva o número 4
6.02
1.602000, 2.670000, 3.140000, 6.020000, Tempo de Execução = 0.282000 ms
```

Programa de ordenação da Questão 14 com implementação dos recursos da biblioteca time.h para medir o tempo de execução da função de ordenação qsort:

```c
#include <stdio.h>      // para o printf
#include <stdlib.h>     // para o qsort e malloc
#include <time.h> //clock(), CLOCKS_PER_SEC e clock_t
int compare (const void * a, const void * b) // função para comparar e
ordenar os números de forma crescente
{
  return ( (int)a - (int)b ); /* foi criado com a seguinte lógica:
  - se a subtração de (int)a e (int)b for menor que zero, significa que o
'b' é maior que o a, e que 'a' irá primeiro que o 'b' na sequência

  - se a subtração de (int)a e (int)b for igual a zero, significa que 'a'
e 'b' são iguais, não importando a ordem deles, ou seja, não muda a
sequência entre um e outro

  - se a subtração de (int)a e (int)b for maior que zero, significa que o
'a' é maior do que o 'b', e que o 'a' irá depois que o 'b' na sequência
*/
}

int main (){ //inicializando a função principal
  clock_t t; //variável para armazenar tempo
  float *x; // criando e definindo o tipo do ponteiro x
  int n, i; // criando variáveis de auxílio

  printf("Digite a quantidade de números que deseja ordenar: \n");
//pedindo que o usuário digite a quantidade desejada de números que quer
ordenar
  scanf("%d", &n);// coletando a entrada fornecida pelo usuário
  x = malloc(n*sizeof(float)); //alocando o vetor/ponteiro e
especificando o tamanho dele dentro da função malloc()
  for(i=0; i<n; i++){// um for() para o programa repetir que o usuário
escreva os números que quer ordenar pela quantidade de vezes necessária,
```

```
informado anteriormente quando pedimos a primeira entrada, armazenada na
variável 'n'
    printf("Escreva o número %d ", i+1);// explicando que o usuário deve
digitar o número (1, 2, 3, ..., n) para a comparação (também poderia ser:
escreva o primeiro número, escreva o segundo número, mas a forma
utilizada ficou mais facilmente automatizada)
    printf("\n");//apenas utilidade visual, pular linha
    scanf("%f", &x[i]);// pedindo uma entrada ao usuário(os números
desejados)
  }
  //Verificando o tempo de execução da função qsort
  t = clock(); //Armazena o tempo
  qsort (x, n, sizeof(float), compare);// dentro da função qsort, foi
indicado, respectivamente o vetor (que usamos de forma alocada), o número
de elementos que serão ordenados, o tamanho de cada elemento e a função
de comparação definida antes do código principal (compare)
  for (i=0; i<n; i++){ // um for() parar repetir o processo de impressão
até que todos os elementos sejam mostrados
     printf ("%f ",x[i]); // aqui solitimos a impressão de todos os
elementos do vetor com apenas duas casas decimais (lembrando que já
estarão todos ordenados em ordem crescente, por consequência da função
qsort() utilizada acima)
  }
  free(x); //foi usado para liberar o ponteiro x, sendo necessário após o
uso do malloc() que está localizado próximo ao início do código
  t = clock() - t; //tempo final - tempo inicial
   printf("Tempo de Execução = %f ms",
((double)t)/((CLOCKS_PER_SEC/1000))); //conversão para double
  return 0; //retornar 0 caso o programa seja executado normalmente
}
```

Output:

```
clang-7 -pthread -lm -o main main.c
./main
Digite a quantidade de números que deseja ordenar:
4
Escreva o número 1
3.14
Escreva o número 2
2.67
Escreva o número 3
1.602
Escreva o número 4
6.02
3.140000 2.670000 1.602000 6.020000 Tempo de Execução = 0.013000 ms
```

Programa de ordenação da Questão 15 com implementação dos recursos da biblioteca time.h para medir o tempo de execução da

função de ordenação feita pelo grupo com recursos da função qsort:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h> //clock(), CLOCKS_PER_SEC e clock_t

float compare(float *ordem2, int n){
    float aux;
    for(int i = 0; i < n; i++){
        for(int j = i; j < n; j++){
            if(ordem2[i] > ordem2[j]){
                aux = ordem2[i];
                ordem2[i] = ordem2[j];
                ordem2[j] = aux;
            }
        }
    }
    return *ordem2;
}

float crescente(float (*px)(float *, int), float *ordem2){

  int n;
    printf("Digite o número de elementos que deseja ordenar: ");
    scanf ("%i", &n);
    ordem2 = (float*) malloc(n*sizeof(float));
    printf("\n");

        for(int i = 0; i < n; i++){
            printf("Digite o elemento %d: \n", i+1);
            scanf("%f", &ordem2[i]);
        }
    px(ordem2, n);

    printf("A ordem crescente dos valores informados segue abaixo: \n");
    for(int i =0; i < n; i++){
            printf("%f  ", ordem2[i]);
    }
    free(ordem2);
    return 0;
}

int main () {
    clock_t t; //variável para armazenar tempo
    float *ordem1;
    //Verificando o tempo de execução da função crescente
    t = clock(); //Armazena o tempo
    crescente(compare, ordem1);
    t = clock() - t; //tempo final - tempo inicial
```

```
    printf("Tempo de Execução = %f ms",
((double)t)/((CLOCKS_PER_SEC/1000))); //conversão para double
    return 0;
}
```

Output:

```
clang-7 -pthread -lm -o main main.c
./main
Digite o número de elementos que deseja ordenar: 4

Digite o elemento 1:
3.14
Digite o elemento 2:
2.67
Digite o elemento 3:
1.602
Digite o elemento 4:
6.02
A ordem crescente dos valores informados segue abaixo:
1.602000  2.670000  3.140000  6.020000  Tempo de Execução = 0.419000 ms
```

Conclusão:

O tempo de execução da função de ordenação da Questão 13 foi 0,282 ms, o da função qsort da Questão 14 foi 0,013 ms e, finalmente, o tempo de execução da função de ordenação da Questão 15 foi 0,419 ms. Portanto, conlui-se que a função de ordenação que possui maior desempenho e eficiência é a função qsort usada no código da Questão 14, uma vez que esta foi a que apresentou o menor tempo de execução.

Referências:

WURTHMANN, George Henrique. Medir tempo de execução em C. wurthmann.blogspot.com, 2015. Disponível em: <http://wurthmann.blogspot.com/2015/04/medir-tempo-de-execucao-em-c.html>. Acesso em: 23/11/2021.