RELATÓRIO ALGORITMOS DE ORDENAÇÃO

ESTRUTURA DE DADOS I

Lucas Hideaki Sakae da Silva

Criação da biblioteca “ordenacao.h”, foram inseridos todas as funções referentes ao vetor e seus métodos de ordenação.

Duas estruturas criadas:

VETOR – recebe os valores do tamanho máximo(‘vetsize’), tamanho usado(‘vetqnt’) e o próprio vetor.
DADO – valores de comparações, trocas e varreduras.

```
struct VETOR
  {
  int vetsize;
  int vetqnt;
  float *vetdata;
  }VETOR;

struct DADOS
  {
  int troca;
  int varredura;
  int comparacao;
  }DADOS;
```

Desempenho(melhor caso)

Bubblesort: posiciona o maior valor sempre na última posição do vetor, assim para cada varredura no vetor, uma posição a menos precisa ser percorrida, o processo se repete até o vetor ficar apenas com duas posições.

```
void bubble(vetor *vet, int size, FILE *arq)
  {
  time_t bub_init, bub_end;
  float bub_tempo;
  int troca = 0, varredura = 0, comparacao = 0;

  bub_init = clock();

  for(int n = 0; n < size - 1; n++)
    {
    for(int m = 0; m < (size - (1 + n)); m++)
      {
      if(vet->vetdata[m] > vet->vetdata[m + 1])
        {
        swap(&vet->vetdata[m], &vet->vetdata[m + 1]);
        troca++;
        }
      comparacao++;
      }
    varredura++;
    }

  bub_end = clock();
  bub_tempo = ((float)(bub_end - bub_init) / CLOCKS_PER_SEC);

  fprintf(arq,"\nBUBBLE ; %d ; %f ; %d ; %d ; %d ", size, bub_tempo, troca, comparacao, varredura);
  printf("\n(TEMPO DE EXECUCAO): %f[s]\n\n", bub_tempo);
  }
```

desempenho O(n^2): quadrático

O(n^2)

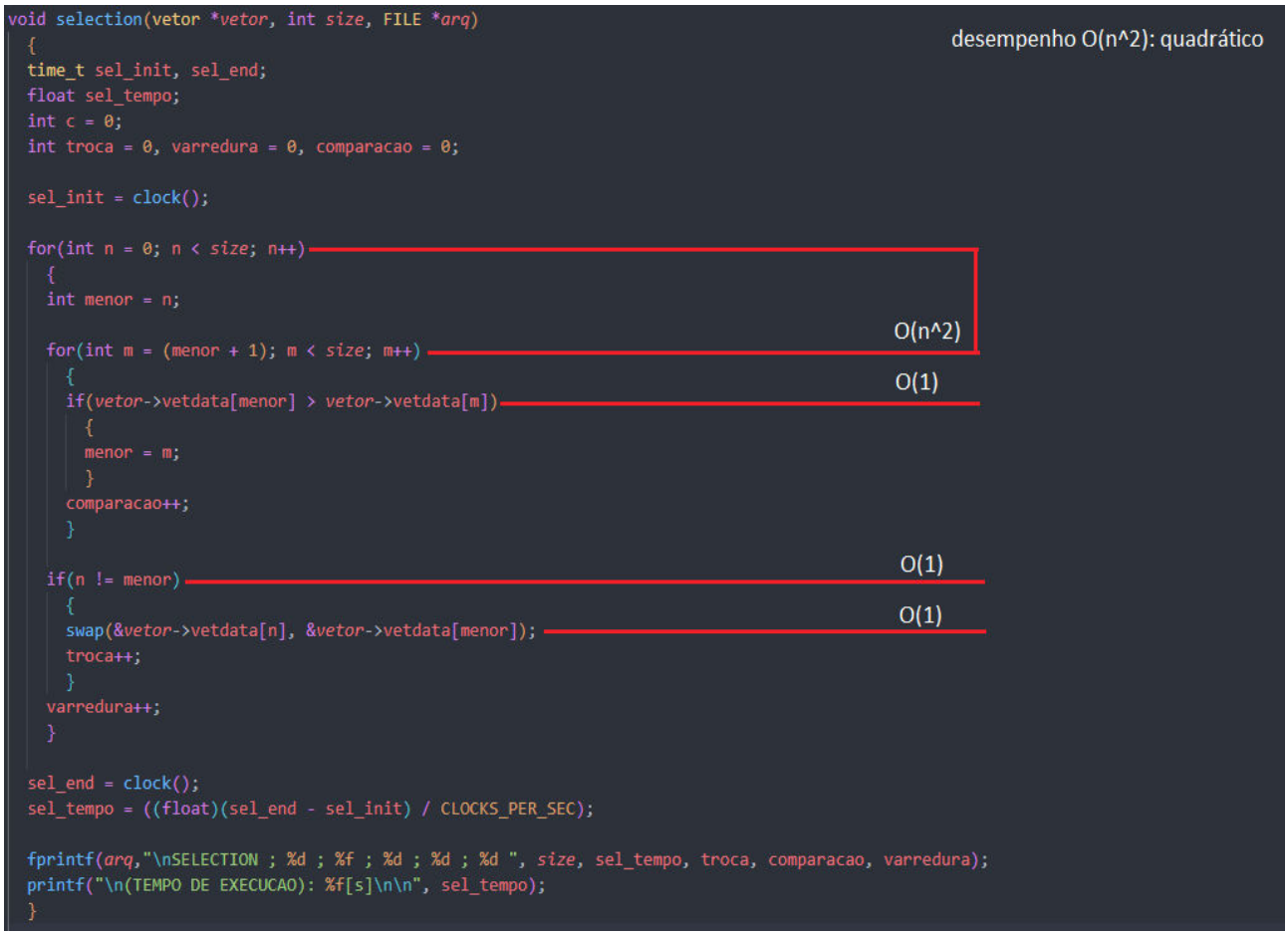
O(1)

O(1)

Processo quadrático possui um desempenho ruim para valores altos, se perde o desempenho conforme o aumento na quantidade de valores de entrada.

Melhoria feita no laço interno ao outro, condição de parada atualiza sempre com uma posição a menos a chamada anterior.

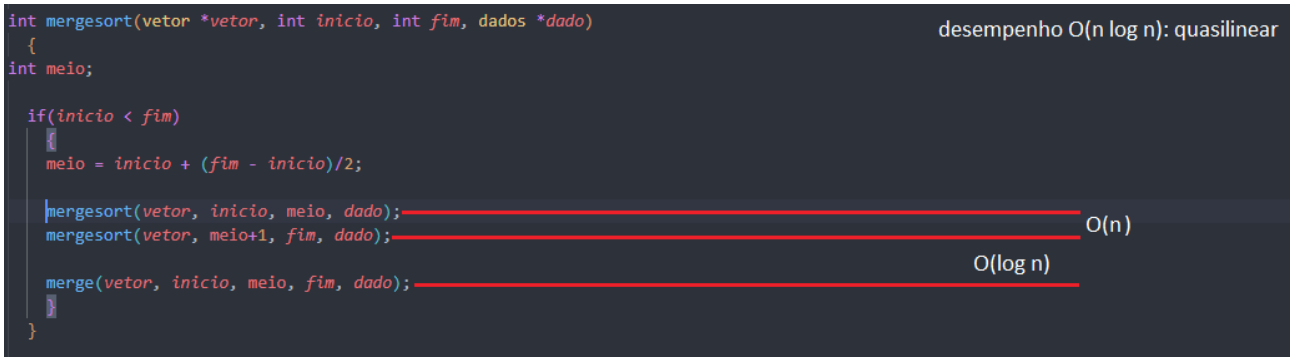
Selectionsort: para cada posição do vetor(fluxo do início ao final), é feita a comparação dessa posição de referência com os outros valores do vetor, caso valor do vetor seja menor, a posição de menor valor é atualizada pela nova posição do vetor, quando termina de percorrer, o vetor troca a referência que armazena a posição para o menor valor com o menor valor encontrado na varredura.




```c
void selection(vetor *vetor, int size, FILE *arq)
  {
  time_t sel_init, sel_end;
  float sel_tempo;
  int c = 0;
  int troca = 0, varredura = 0, comparacao = 0;

  sel_init = clock();

  for(int n = 0; n < size; n++)
    {
    int menor = n;

    for(int m = (menor + 1); m < size; m++)
      {
      if(vetor->vetdata[menor] > vetor->vetdata[m])
        {
        menor = m;
        }
      comparacao++;
      }

    if(n != menor)
      {
      swap(&vetor->vetdata[n], &vetor->vetdata[menor]);
      troca++;
      }
    varredura++;
    }

  sel_end = clock();
  sel_tempo = ((float)(sel_end - sel_init) / CLOCKS_PER_SEC);

  fprintf(arq,"\nSELECTION ; %d ; %f ; %d ; %d ; %d ", size, sel_tempo, troca, comparacao, varredura);
  printf("\n(TEMPO DE EXECUCAO): %f[s]\n\n", sel_tempo);
  }
```

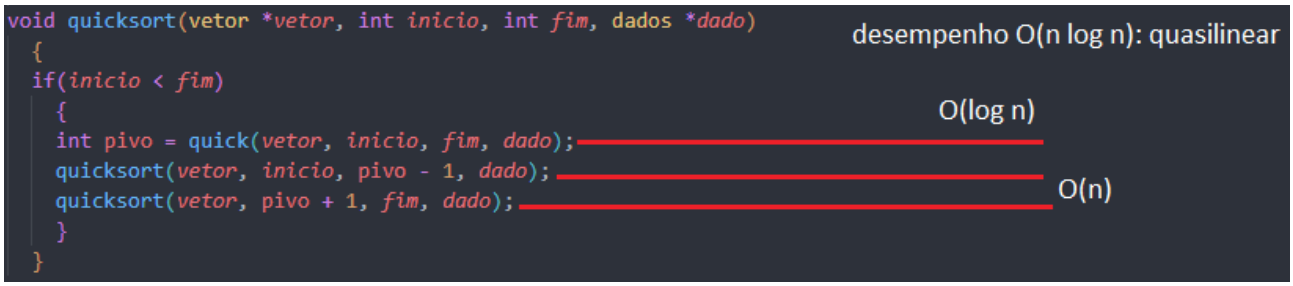
Função quadrática, melhoria feita no laço interno, a posição de inicialização da varredura sempre começa uma posição a frente do que já foi colocado o menor valor anterior, percorrendo sempre uma posição a menos para cada chamada no laço.

Mergesort: o “mergesort” é composto por duas funções e trabalha com recursividade, a função “mergesort” separa o vetor em dois até que possua apenas 1 posição, assim quando a recursividade for acabando a função “merge” é chamada para realizar a ordenação das metades daquela instância. Dentro do “merge” dois vetores de referência são criados, um para cada metade do vetor principal, essas duas referências são comparadas e os menores valores entre as metades vão sendo armazenadas dentro do vetor principal unindo os dois vetores em ordem.



A função ‘mergesort’ realiza a divisão do vetor de forma recursiva, dependendo apenas do tamanho do vetor recebido O(n) e o ‘merge’ realiza a ordenação de forma O(log n), onde o crescimento do número de execuções é menor que o crescimento do tamanho dos dados de entrada.

Quicksort: para a ordenação do “quick”, o processo é contrário ao “merge”, o “quick” separa o vetor em dois utilizando um pivo(definido como a primeira posição no código), duas referências de posição são criadas, uma no início e outra no fim, a primeira referência percorrerá o vetor até que o valor dela seja maior que o pivo, assim guardaremos a posição em que esse valor está, o mesmo processo sera feito com o lado do fim, porém a posição do vetor irá diminuir até que encontre um valor menor que o pivo(fazendo busca oposta a outra referência), quando ambos processos terminam, seus valores são trocados e novamente é realizado a comparação entre o pivo e as duas referências, quando as posições das referências se cruzam a posição do pivo é definida(posição correta no vetor principal).
Devido a recursão, esse processo se repete até que todos os pivos estejam nas suas posições corretas e a divisão do vetor chegue no fim.



Melhorias que poderiam ser feitas:
- escolha do pivo(para uma divisão mais eficiente)
- verificação de ordenação(vetor já ordenado)

Obs.: Erros de execução para vetores com tamanho acima de 100000.

MÉTODO	TAMANHO	EXECUÇÃO(s)	TROCA	COMPARAÇÃO	VARREDURA
BUBBLE	1000	0,008000	253384	499500	999
SELECTION	1000	0,003000	995	499500	1000
MERGE	1000	0,000000	9976	8715	999
QUICK	1000	0,000000	erro	erro	erro
BUBBLE	10000	0,547000	25341218	49995000	9999
SELECTION	10000	0,171000	9993	49995000	10000
MERGE	10000	0,015000	133616	120428	9999
QUICK	10000	0,015000	9214	3717901	erro
BUBBLE	100000	64,014999	1788922485	704982704	99999
SELECTION	100000	16,586000	99881	704982704	100000
MERGE	100000	0,062000	1668928	1536134	99999
QUICK	100000	0,026000	318978	2300627	erro
BUBBLE	1000000	7303,703125	1014916430	1783293664	999999
SELECTION	1000000	1757,917969	999952	1783293664	1000000
MERGE	1000000	0,658000	19951424	18673530	999999