UFRR

UNIVERSIDADE FEDERAL DE RORAIMA

Algoritmos de ordenação

Alunos: Cristiano Lima / João Roberto

BOA VISTA, RR

2019

Selection sort

Consiste em ordenar a lista “selecionando” a cada iteração o menores itens possíveis e os colocam da esquerda para a direita.

```
void selecao (int vet, int tam){
    int i, j, min, x;
    for (i=1; i<=n-1; i++){
        min = i;
    for (j=i+1; j<=n; j++){
            if (vet[j] < vet[min])
            min = j;
    }
    x = vet[min];
    vet[min] = vet[i];
    vet[i] = x;
    }
}
```

Desvantagem

Ele é um dos mais lentos para vetores de tamanhos grandes.

Ele não é estável.

Vantagem

Não necessita de um vetor auxiliar

Por não usar um vetor auxiliar para realizar a ordenação, ele ocupa menos memória.

Ele é uns dos mais velozes na ordenação de vetores de tamanhos pequenos.

Insertion sort

é o algoritmo de ordenação que, dado uma estrutura (array, lista) constrói uma matriz final com um elemento de cada vez, uma inserção por vez.

```
int k, j, aux;

for (k = 1; k <= n - 1; k++){
   printf("\n[%d] ", k);

   aux = vetor[k];
   j = k - 1;
   while (j >= 0 && aux < vetor[j]
      printf("%d, ", j);

      vetor[j+1] = vetor[j];
      j--;
   }

   vetor[j+1] = aux;
}
```

Desvantagem

Alto custo de movimentação de elementos no vetor.

Vantagem

É um bom método quando se desejar adicionar poucos elementos em um arquivo já ordenado, pois seu custo é linear.

O algoritmo de ordenação por inserção é estável.

Bubble sort

Percorrer o vetor diversas vezes, e a cada passagem fazer flutuar para o topo o maior elemento da sequência

```c
void bubble_sort (int vetor[], int n) {
    int k, j, aux;

    for (k = 1; k < n; k++) {
        printf("\n[%d] ", k);

        for (j = 0; j < n - 1; j++) {
            printf("%d, ", j);

            if (vetor[j] > vetor[j + 1]) {
                aux          = vetor[j];
                vetor[j]     = vetor[j + 1];
                vetor[j + 1] = aux;
            }
        }
    }
}
```

Desvantagem

Muito lento

Vantagem

Os elementos são trocados sem utilizar armazenamento temporário

Radix sort

É um algoritmo de ordenação que ordena inteiros processando dígitos individuais.

```c
void radixsort(int vetor[], int tamanho) {
    int i;
    int *b;
    int maior = vetor[0];
    int exp = 1;

    b = (int *)calloc(tamanho, sizeof(int));

    for (i = 0; i < tamanho; i++) {
        if (vetor[i] > maior)
            maior = vetor[i];
    }

    while (maior/exp > 0) {
        int bucket[10] = { 0 };
        for (i = 0; i < tamanho; i++)
            bucket[(vetor[i] / exp) % 10]++;
        for (i = 1; i < 10; i++)
            bucket[i] += bucket[i - 1];
        for (i = tamanho - 1; i >= 0; i--)
            b[--bucket[(vetor[i] / exp) % 10]] = vetor[i];
        for (i = 0; i < tamanho; i++)
            vetor[i] = b[i];
        exp *= 10;
    }

    free(b);
}
```

Vantagem

Estável

Não compara as chaves

QuickSort

é um método de ordenação muito rápido e eficiente. A estratégia consiste em rearranjar as chaves de modo que as chaves "menores" precedam as chaves "maiores".

```
void Quick(int vetor[10], int inicio, int fim){

   int pivo, aux, i, j, meio;

   i = inicio;
   j = fim;

   meio = (int) ((i + j) / 2);
   pivo = vetor[meio];

   do{
      while (vetor[i] < pivo) i = i + 1;
      while (vetor[j] > pivo) j = j - 1;

      if(i <= j){
         aux = vetor[i];
         vetor[i] = vetor[j];
         vetor[j] = aux;
         i = i + 1;
         j = j - 1;
      }
   }while(j > i);

   if(inicio < j) Quick(vetor, inicio, j);
   if(i < fim) Quick(vetor, i, fim);

}
```

Desvantagem

Não é estável

Pior caso é quadrático

Vantagem

Melhor opção para ordenar vetores grandes Muito rápido por que o laço interno é simples

Memória auxiliar para a pilha de recursão é pequena

Merge Sort

A ideia do Merge Sort é dividir o vetor em dois subvetores, cada um com metade dos elementos do vetor original. Esse procedimento é então reaplicado aos dois subvetores recursivamente. Quando os subvetores têm apenas um elemento (caso base), a recursão para. Então, os subvetores ordenados são fundidos (ou intercalados) num único vetor ordenado.

```c
void merge(int vetor[], int comeco, int meio, int fim) {
    int com1 = comeco, com2 = meio+1, comAux = 0, tam = fim-comeco+1;
    int *vetAux;
    vetAux = (int*)malloc(tam * sizeof(int));

    while(com1 <= meio && com2 <= fim){
        if(vetor[com1] < vetor[com2]) {
            vetAux[comAux] = vetor[com1];
            com1++;
        } else {
            vetAux[comAux] = vetor[com2];
            com2++;
        }
        comAux++;
    }

    while(com1 <= meio){  //Caso ainda haja elementos na primeira metade
        vetAux[comAux] = vetor[com1];
        comAux++;
        com1++;
    }

    while(com2 <= fim) {   //Caso ainda haja elementos na segunda metade
        vetAux[comAux] = vetor[com2];
        comAux++;
        com2++;
    }

    for(comAux = comeco; comAux <= fim; comAux++){    //Move os elementos de volta para o vetor original
        vetor[comAux] = vetAux[comAux-comeco];
    }

    free(vetAux);
}
```

Desvantagem

Utiliza funções recursivas;

Gasto extra de memória.

Vantagem

Requerem um número menor de acessos à memória.

Heap sort

o heapsort usa um Heap, Heap é uma árvore binária com as seguintes propriedades:

o pai tem que ser maior que os filhos

A árvore é perfeitamente balanceada e as folhas no último nível estão todas nas posições mais a esquerda.

```
void heapsort(int a[], int n) {
   int i = n / 2, pai, filho, t;
   while(true) {
      if (i > 0) {
          i--;
          t = a[i];
      } else {
          n--;
          if (n <= 0) return;
          t = a[n];
          a[n] = a[0];
      }
      pai = i;
      filho = i * 2 + 1;
      while (filho < n) {
          if ((filho + 1 < n)  &&  (a[filho + 1] > a[filho]))
              filho++;
          if (a[filho] > t) {
             a[pai] = a[filho];
             pai = filho;
             filho = pai * 2 + 1;
          } else {
             break;
          }
      }
      a[pai] = t;
   }
}
```

Desvantagem

construção constante de árvores

Vantagem

Comportamento O(n lg n) no pior caso

Busca Sequencial

Tipo de pesquisa em vetores ou listas de modo sequencial, elemento por elemento, de modo que a função do tempo em relação ao número de elementos é linear, ou seja, cresce proporcionalmente.

```c
int procura(char vetor[], int tamanho, char elementoProcurado) {
    int i;
    for (i = 0; i < tamanho; i++) {
        if (vetor[i] == elementoProcurado) {
            return i;
        }
    }

    return -1;
}
```

Desvantagem

o seu pior caso seria O(n)

Vantagem

É ideal para um arranjo não ordenado

Busca Binária

Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca (Divisão e conquista) comparando o elemento buscado (chave) com o elemento no meio do vetor

```
int PesquisaBinaria ( int k[], int chave , int N)
{
 int inf,sup,meio;
 inf=0;
 sup=N-1;
 while (inf<=sup)
 {
      meio=(inf+sup)/2;
      if (chave==k[meio])
           return meio;
      else if (chave<k[meio])
           sup=meio-1;
      else
           inf=meio+1;
 }
 return -1;   /* não encontrado */
}
```

Desvantagem

Nem todo arranjo está ordenado

Vantagem

Eficiência da busca

https://medium.com/@henriquebraga_18075/algoritmos-de-ordena%C3%A7%C3%A3o-ii-selection-sort-8ee4234deb10

https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/

http://www.devfuria.com.br/logica-de-programacao/exemplos-na-linguagem-c-do-algoritmo-bubble-sort/

http://cafofodoprogramador.blogspot.com/2009/02/busca-binaria-em-linguagem-c.html
http://www.devfuria.com.br/logica-de-programacao/introducao-ao-algoritmo-de-ordenacao-insertion-sort/

https://mundobitabitblog.wordpress.com/2017/07/03/101/

http://www.inf.ufg.br/~hebert/disc/aed1/AED1_05_ordenacao2.pdf

https://www.blogcyberini.com/2018/07/merge-sort.html

https://pt.wikipedia.org/wiki/Merge_sort#C%C3%B3digo_em_C

http://www.ufjf.br/jairo_souza/files/2009/12/2-Ordena%C3%A7%C3%A3o-HeapSort.pdf

https://pt.wikipedia.org/wiki/Heapsort

https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/heapsort.pdf

https://www.mundojs.com.br/2018/02/05/algoritmos-de-busca-sequencial-e-binaria/#page-content
https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/quicksort.pdf
https://www.vivaolinux.com.br/script/Ordenacao-QuickSort