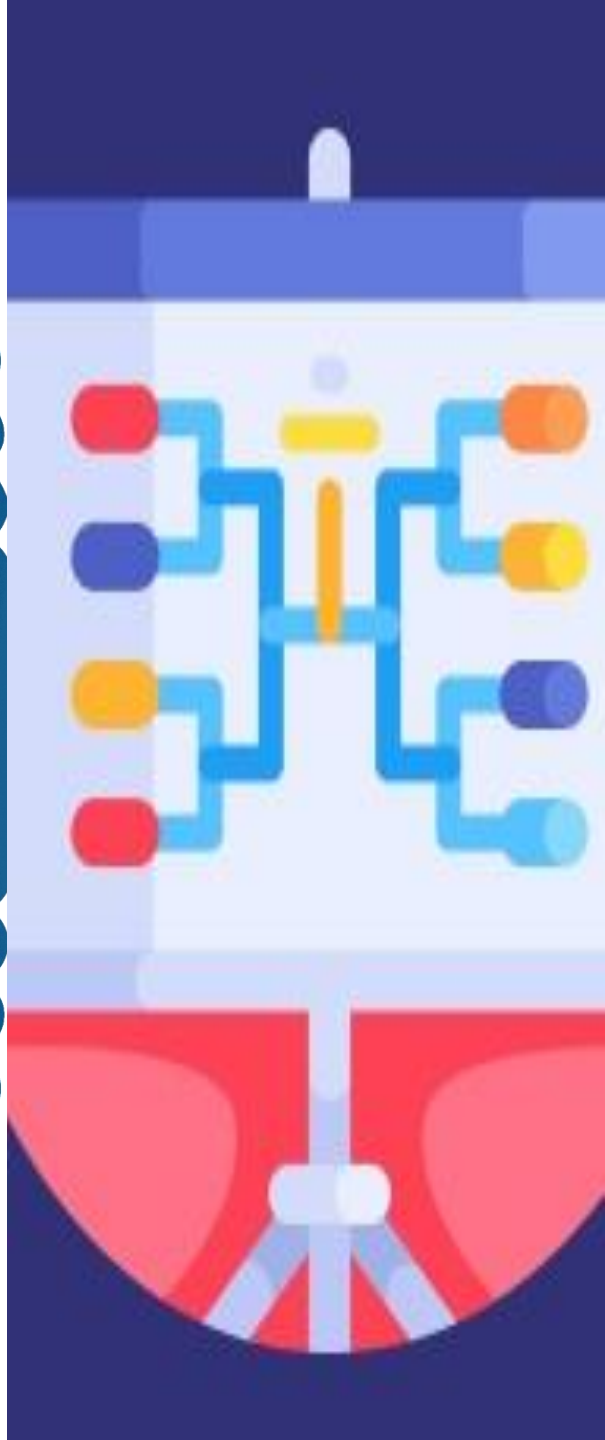
Unicesumar

Estrutura de Dados II



Unidade 7 – ALGORITMOS DE ORDENAÇÃO AVANÇADOS

Prof. Sandro T. Pinto

Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou de objetos em uma ordem ascendente ou descendente. A ordenação visa facilitar a recuperação A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado

http://www2.dcc.ufmg.br/livros/algoritmos/

unidade 7

ALGORITMOS DE ORDENAÇÃO AVANÇADOS - Introdução

Na aula anterior, estudamos quatro algoritmos de ordenação. Os dois primeiros, Bubblesort e Selectionsort, eram de implementação mais simplificada. O terceiro, Insertionsort, ligeiramente mais complexo, utiliza um laço de repetição interno para tentar otimizar a ordenação e finalmente o Shellsort, que se apropria do Insertionsort para realizar a ordenação de um vetor considerando o conceito de gaps.

6 5 3 1 8 7 2 4

unidade 7

ALGORITMOS DE ORDENAÇÃO AVANÇADOS- Introdução

Nesta aula veremos mais três técnicas de ordenação, sua complexidade será crescente, assim com o seu desempenho.

Ordenação por Mergesort

Ordenação por Quicksort

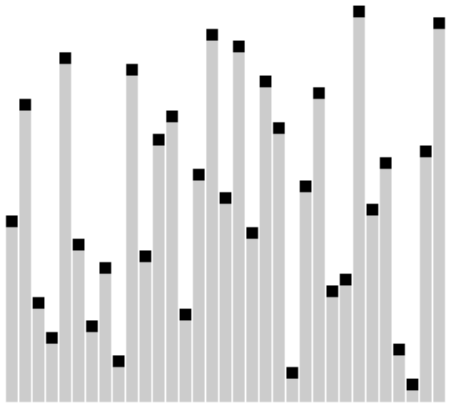
Ordenação por Heapsort

unidade 7

Técnicas de Ordenação - Mergesort

A técnica de ordenação Mergesort utiliza um conceito conhecido por dividir para conquistar. Esse conceito sugere que um problema complexo possa ser dividido em dois problemas menores, e cada um desses sejam divididos novamente em partes menores ainda, até que se encontre uma parte pequena e simples suficiente para que seja resolvido.

6 5 3 1 8 7 2 4



unidade 7

Técnicas de Ordenação - Mergesort

O algoritmo Mergesort faz isso de forma recursiva(repetida). Assim que o vetor é dividido, cada uma das metades é passada como parâmetro a uma nova chamada da função Mergesort. Essa recursividade desce até o ponto em que o vetor tem apenas um único valor. Nesse momento, inicia-se o retorno da recursividade, e os vetores unitários são comparados e unidos já ordenados.

Dança Mergesort

unidade 7

Técnicas de Ordenação - Mergesort

O Programa traz uma implementação em linguagem C da técnica de ordenação Mergesort.

A primeira função é bem simples e mostra a recursividade da técnica. Ela recebe como parâmetro o vetor vec a ser ordenado, o tamanho tam do vetor e uma variável qtd inteira usada pra medir o esforço computacional do algoritmo

```
//Aplica o modo mergeSort
int mergeSort(int vec[], int tam, int qtd) {
 int meio;
 if (tam > 1) {
 meio = tam / 2;
 qtd = mergeSort(vec, meio, qtd);
 qtd = mergeSort(vec + meio, tam - meio, qtd);
 junta(vec, tam);
 }
 return (qtd+1);
}
//Junta os pedaços num novo vetor ordenado
void junta(int vec[], int tam) {
 int i, j, k;
 int meio;
 int* tmp;
 tmp = (int*) malloc(tam * sizeof(int));
 if (tmp == NULL) {
 exit(1);
 }
```

unidade 7

Técnicas de Ordenação - Mergesort

Se o tamanho do vetor for maior do que um, o programa procura o meio do vetor e aplica a recursão duas vezes, uma para o início até a metade do vetor atual e outra da metade até o final do vetor.

Depois que o vetor for transformado em partes unitárias, a recursividade volta chamando a função junta. Ela irá verificar o valor das partes antes de realizar a junção de forma ordenada.

```
meio = tam / 2;
i = 0;
j = meio;
k = 0;
while (i < meio && j < tam) {
if (vec[i] < vec[j]) {
tmp[k] = vec[i];
++i;
}
else {
tmp[k] = vec[j];
++j;
}
++k;
}
if (i == meio) {
while (j < tam) {
tmp[k] = vec[j];
++j;
++k;
}
}
else {
while (i < meio) {
tmp[k] = vec[i];
++i;
++k;
}
}
for (i = 0; i < tam; ++i) {
vec[i] = tmp[i];
}
free(tmp);
}
```

unidade 7

Técnicas de Ordenação - Mergesort

A primeira coisa que o algoritmo faz é dividir o vetor em dois a aplicar recursividade em cada uma das metades:

0	1	2	3	4
3	1	8	7	20

5	6	7	8	9
21	31	40	30	0

Cada uma das chamadas ao Mergesort irá dividir novamente o vetor, recursivamente.

0	1
3	1

2	3	4
8	7	20

5	6	7
21	31	40

8	9
30	0

unidade 7

Técnicas de Ordenação - Mergesort

O processo se repete até que o cada vetor contenha apenas um valor.

0	1	2	3	4	5	6	7	8	9
3	1	8	7	20	21	31	40	30	0

Nesse momento não há mais chamadas recursivas e começa o retorno para a chamada original aplicando a função junta nos pares de vetores, já ordenados.

0	1	2	3	4	5	6	7	8	9
1	3	8	7	20	21	31	40	0	3

unidade 7

Técnicas de Ordenação - Mergesort

E o procedimento se repete até que tenhamos apenas um vetor e o mesmo se encontrará totalmente ordenado.

0	1	2	3	4	5	6	7	8	9
0	1	3	7	8	20	21	30	31	40

O Mergesort foi criado em 1945 pelo matemático húngaro chamado John Von Newmann. Apesar de apresentar bom desempenho em vetores não muito grandes, sua implementação e ideia são complexos se comparado com o Bubblesort e Selectionsort.

unidade 7

Técnicas de Ordenação - QUICKSORT

A segunda técnica de ordenação que veremos nesta unidade é o Quicksort.

Segundo Cormen (2012), esse método também é conhecido por classificação por troca de partição. Criado em 1960 pelo cientista da computação britânico Sr. Charles Antony Richard Hoare, ele é considerado o algoritmo de ordenação mais utilizado no mundo. Sua publicação ocorreu em 1962 após uma série de refinamentos.

6 5 3 1 8 7 2 4

unidade 7

Técnicas de Ordenação - QUICKSORT

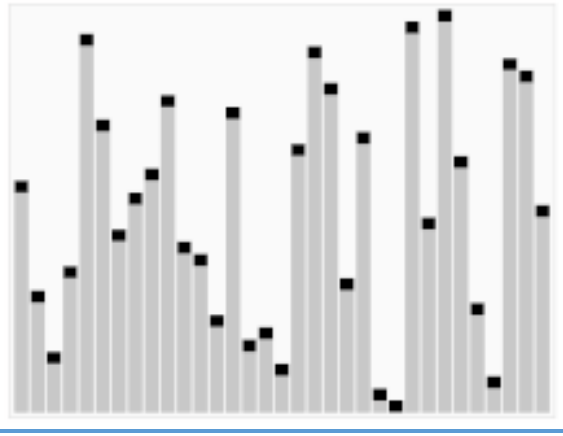
Essa técnica também utiliza a estratégia de dividir para conquistar. O primeiro passo é escolher um elemento qualquer que será denominado de pivô. A partir desse elemento, a lista será dividida em três sublistas, uma para o pivô, uma para os valores menores e outra para os valores maiores do que o próprio pivô.

Dança do Quicksort

unidade 7

Técnicas de Ordenação - QUICKSORT

Isso garante que as chaves menores precedam as chaves maiores e que o pivô esteja na sua correta posição dentro do vetor. Essa técnica é muito parecida com a árvore de busca binária. As duas sublistas (partições) ainda não ordenadas são chamadas de forma recursiva até que cada uma das inúmeras sublistas criadas tenha apenas um elemento e o vetor se encontre ordenado.



unidade 7

Técnicas de Ordenação - QUICKSORT

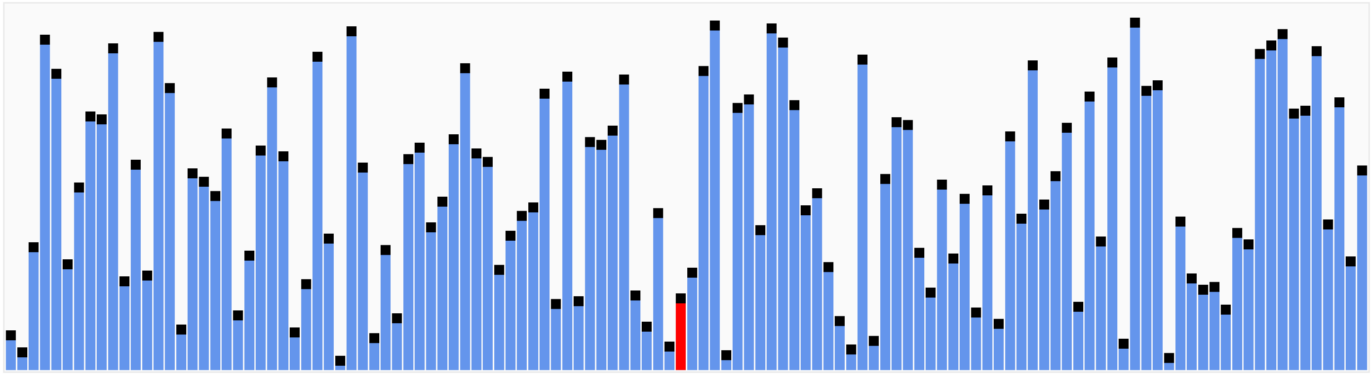
O Programa apresenta duas funções. A primeira é o Quicksort propriamente dito a sua chamada recursiva. A cada iteração ele invoca a função Particiona, que vai escolher o pivô e criar duas novas listas a serem ordenadas.

```
//Aplica o modo do quickSort
int quickSort(int vec[], int left, int right, int qtd) {
 int r;
 if (right > left) {
 r = particiona(vec, left, right);
 qtd = quickSort(vec, left, r - 1, qtd);
 qtd = quickSort(vec, r + 1, right, qtd);
 }
 return (qtd +1);
}
//Divide o vetor em pedacos menores
int particiona(int vec[], int left, int right) {
 int i, j;
 i = left;
 for (j = left + 1; j <= right; ++j) {
 if (vec[j] < vec[left]) {
 ++i;
 troca(&vec[i], &vec[j]);
 }
 }
 troca(&vec[left], &vec[i]);
 return i;
}
```

unidade 7

Técnicas de Ordenação - QUICKSORT

Esse algoritmo também se assemelha ao Mergesort. A principal diferença é que o Quicksort trabalha com um pivô numa posição aleatória e, durante o processo de partição, o pivô já estará na sua posição final do vetor. O Mergesort divide a estrutura sempre pela metade e inicia o processo de ordenação apenas no final do processo durante o retorno da recursividade.



unidade 7

Técnicas de Ordenação - QUICKSORT

Vamos fazer uma simulação do Quicksort no vetor vec desordenado:

0	1	2	3	4	5	6	7	8	9
3	1	8	7	20	21	31	40	30	0

Qualquer elemento pode ser escolhido como pivô. Escolhi começar por vec[0]=3. Vamos separar a lista agora em três partes, uma com o pivô, uma com os elementos menores que 3 e outra com elementos maiores que 3.

0	1	2	3	4	5	6	7	8	9
1	0	3	8	7	20	21	31	40	30

Técnicas de Ordenação - QUICKSORT

O valor escolhido para o pivô (3) já se encontra na sua devida posição na lista, e à sua esquerda está a sub lista com valores menores que 3 e à direita outra sub lista com valores maiores que 3.

Aplicaremos a recursividade em cada uma dessas sub listas. Para ficar mais claro o entendimento, vamos tratar as duas chamadas recursivas separadamente, primeiro a da sub lista com valores menores que o pivô.

unidade 7

Técnicas de Ordenação - QUICKSORT

Escolheremos nela um elemento qualquer para ser o novo pivô na recursão. Vamos pegar vec[0]=1.

Os valores da sub lista serão divididos novamente, ficando os valores menores à esquerda e os maiores à direita.

0	1	2	3	4	5	6	7	8	9
0	1	3	8	7	20	21	31	40	30

unidade 7

Técnicas de Ordenação - QUICKSORT

O valor do pivô (1) já se encontra na sua devida posição na lista. Como sobrou apenas um elemento na sub lista (0), o mesmo já se encontra ordenado. Agora vamos tratar da recursão do outro lado do primeiro pivô.

Faremos diferente e vamos escolher vec[9]=30 como novo pivô. Dividiremos a lista em duas sub listas e aplicaremos novamente a recursão. Uma das listas terá apenas valores menores do que 30 e a outra apenas valores maiores.

0	1	2	3	4	5	6	7	8	9
0	1	3	8	7	20	21	30	31	40

unidade 7

Técnicas de Ordenação - QUICKSORT

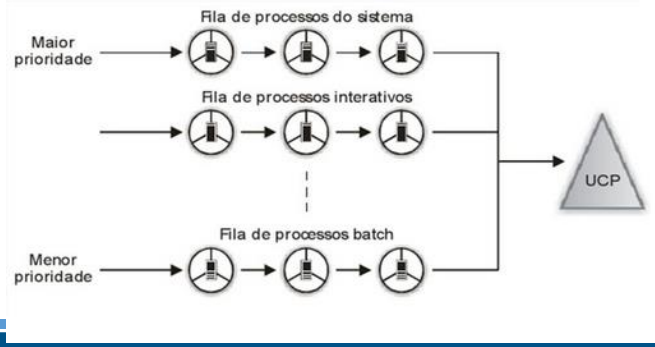
O algoritmo ainda não sabe, mas a parte superior da lista já se encontra ordenada. Mesmo assim aquela parte também sofrerá recursão e em mais uma interação estará pronta. A sub lista com os valores menores também está quase ordenada, e a quantidade de passos necessários para a finalização depende da escolha do pivô. Se for escolhido 7 ou 8, o vetor já ficará ordenado. Se for escolhido 20 ou 21, será necessário ainda mais uma iteração para encontrar o vetor original devidamente ordenado.

0	1	2	3	4	5	6	7	8	9
0	1	3	8	7	20	21	30	31	40

unidade 7

Técnicas de Ordenação - HEAPSORT

Para compreender como o Heapsort realiza a ordenação de um arranjo, devemos remeter a outra estrutura de dados: as filas de prioridade. Uma fila de prioridades agrupa elementos de forma que cada um dos elementos pode ter maior ou menor importância para a aplicação. Em suma, nesse tipo de fila é possível inserir elementos a qualquer instante e em qualquer posição do arranjo, de acordo com sua prioridade. Já a remoção é sempre feita no elemento de maior prioridade.



unidade 7

Técnicas de Ordenação - HEAPSORT

A implementação de uma fila de prioridades eficiente advém da estrutura de dados heap(pilha). Uma heap permite a inserção e remoção de elementos em filas de prioridade em tempo logarítmico, o que é algo bastante eficiente. Tamanha eficiência é alcançada a partir da transformação de um vetor linear em uma estrutura similar a uma árvore binária. Todavia devemos lembrar que o algoritmo Heapsort não implementa uma fila de prioridades, ou seja, são coisas distintas.

Dança do Heapsort

unidade 7

Técnicas de Ordenação - HEAPSORT

Podemos definir a estrutura de dados heap como uma árvore binária com algumas propriedades adicionais. Considere uma árvore binária com N níveis, que vão de 0 até N-1:

- A heap deve ser uma árvore binária eficiente, por isso é preciso que ela seja uma árvore completa até o nível N-2. Isto é, a heap é, obrigatoriamente, uma árvore binária completa até o penúltimo nível.
- Por convenção a heap deve fazer com que os nós do nível N-1 (último nível) estejam tão à esquerda quanto possível

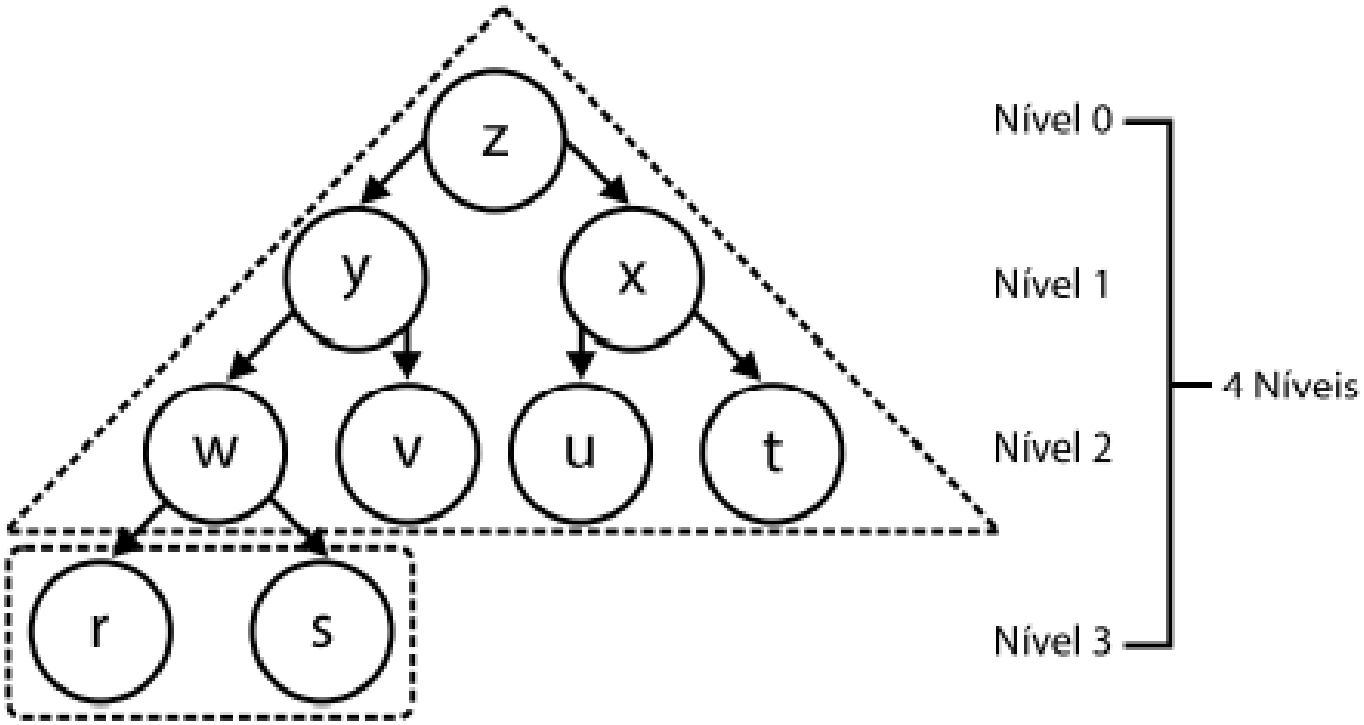
unidade 7

Técnicas de Ordenação - HEAPSORT

- A chave de cada nó deve ser comparada ao seu nó pai. Ou seja, o conteúdo de nós x e y, cujas sub árvores são enraizadas em z, devem respeitar a seguinte regra:
- No caso de uma max-heap, o nó raiz deve ser maior ou igual aos nós filhos x e y.
- Já em uma min-heap, o nó raiz deve ser menor ou igual aos nós filhos x e y

unidade 7

Técnicas de Ordenação - HEAPSORT



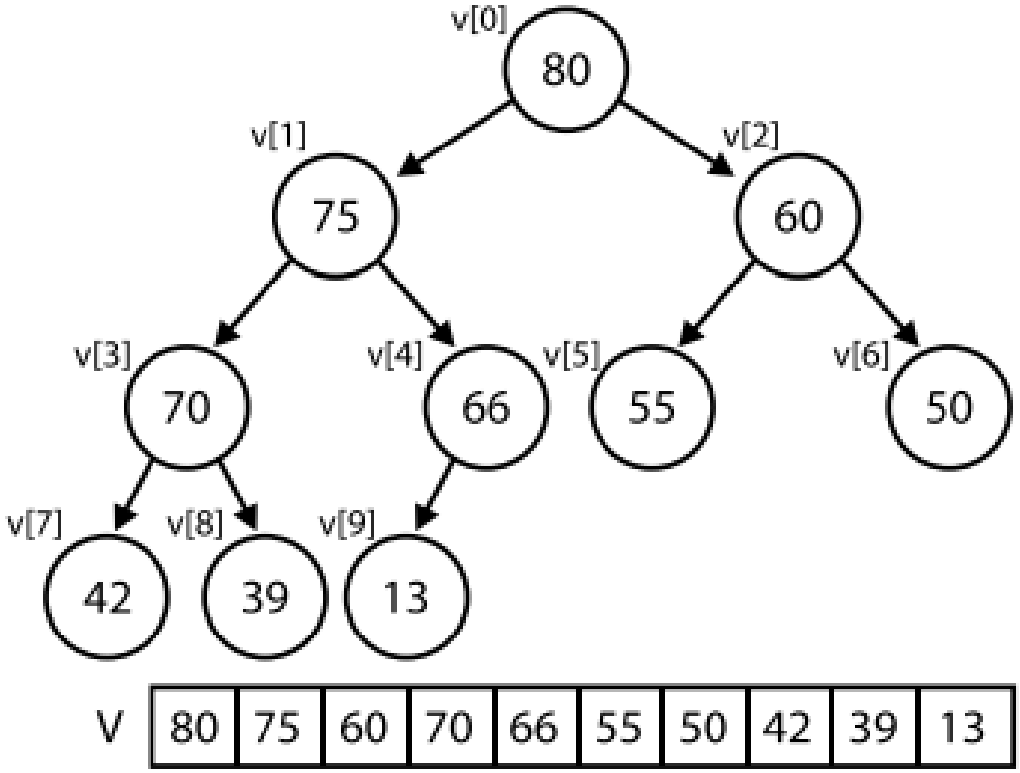
unidade 7

Técnicas de Ordenação - HEAPSORT

Tais propriedades nos auxiliam a armazenar a heap em um vetor, ao invés de ter de trabalhar com alocação dinâmica de memória. Apenas para relembrar: se um nó pai está na posição p do vetor, então seu filho esquerdo estará na posição 2*p+1 e seu filho direito na posição 2*p+2. Dessa forma, observe como a heap representada visualmente na Figura pode ser armazenada em um vetor v.

unidade 7

Técnicas de Ordenação - HEAPSORT



unidade 7

Técnicas de Ordenação - HEAPSORT

Repare que as propriedades da heap garantem um fato importante: o maior elemento entre todos sempre estará armazenado na raiz, isto é, na posição inicial do vetor (v[0]). Dessa forma, podemos pensar em um algoritmo para se aproveitar dessa característica para realizar a ordenação em um vetor. Daremos a esse algoritmo o nome de Heapsort.

unidade 7

Técnicas de Ordenação - HEAPSORT

Primeiramente, precisamos garantir que o vetor esteja formatado como uma heap, de acordo com as fórmulas de posicionamento apresentadas anteriormente. Damos o nome de constroiHeap ao método que realiza essa façanha (em inglês, Build-Max-Heap). Além de construir uma árvore binária quase completa dentro do vetor, o método constroiHeap é responsável por garantir que cada nó pai seja maior ou igual aos nós filhos.

unidade 7

Técnicas de Ordenação - HEAPSORT

Em seguida, devemos nos concentrar nas extremidades do vetor de forma a considerar que, conforme o Heapsort vai sendo executado, nas partes iniciais do vetor, temos os dados da heap, e nas partes finais do vetor, temos o arranjo ordenado. Em suma, durante o processo de ordenação, dividimos o vetor logicamente em duas porções: a heap e a porção ordenada do vetor.

unidade 7

Técnicas de Ordenação - HEAPSORT

Uma vez que o vetor desordenado foi transformado em heap, podemos dar sequência. Na primeira posição do vetor (raiz da heap), temos o maior elemento de todos. Se nossa intenção é ordenar o vetor em ordem não-decrescente (de modo geral, crescente), podemos simplesmente trocar o maior elemento da raiz pelo elemento que se encontra ao final da heap.

unidade 7

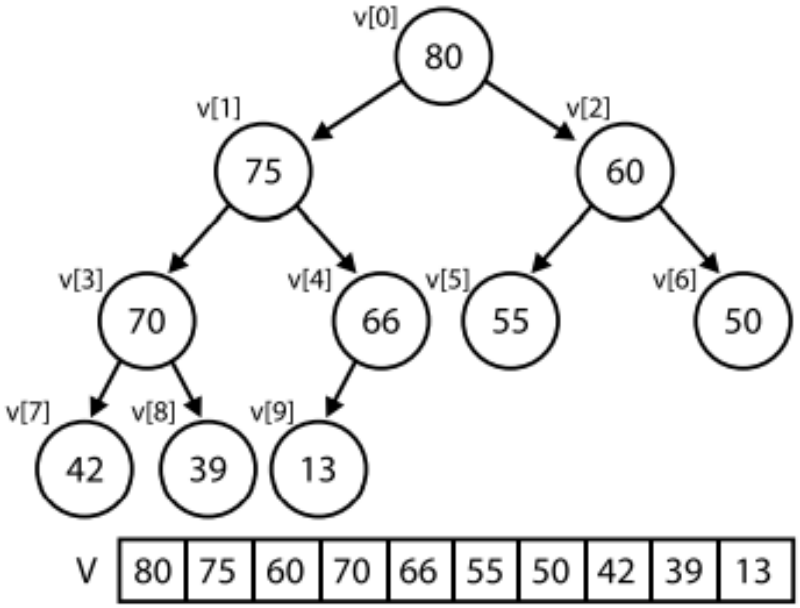
Técnicas de Ordenação - HEAPSORT

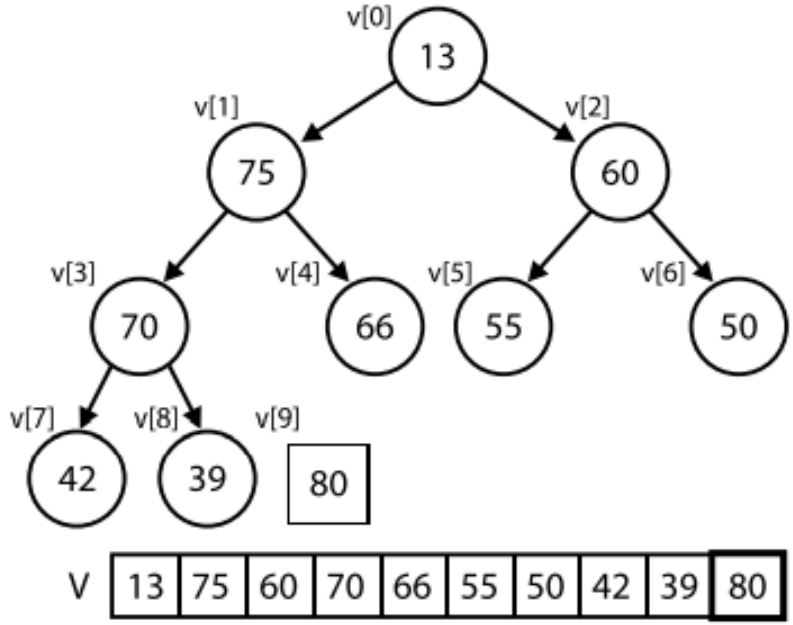
Quando trocamos o elemento da raiz da heap com o elemento do final do vetor, estamos posicionando o maior elemento em sua posição ordenada final. Nesse instante, devemos desconsiderar tal elemento como um nó da heap de forma que, agora, ele passe a pertencer à porção ordenada do vetor.

unidade 7

Técnicas de Ordenação - HEAPSORT

Observe como o nó 80, raiz do vetor v da Figura à esquerda, foi trocado com o nó de chave igual a 13, resultando na Figura à direita.





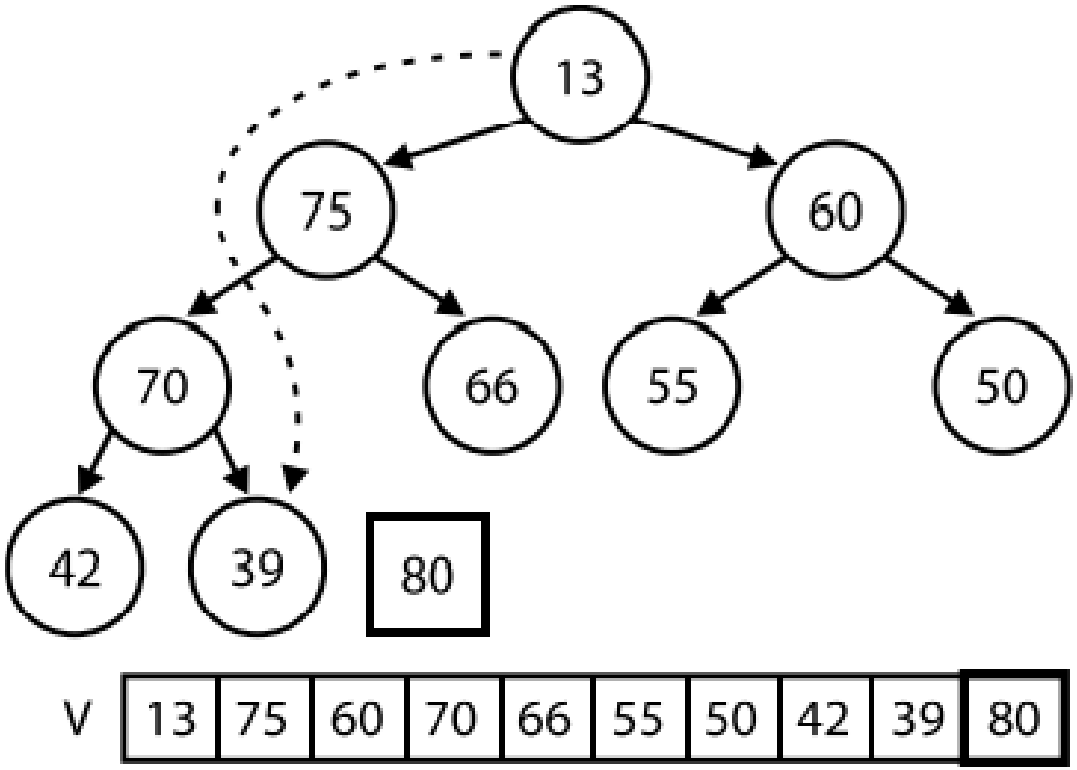
unidade 7

Técnicas de Ordenação - HEAPSORT

Observe que o elemento 80, de fato, é o maior de todos e, após a troca, foi posicionado no último índice de v. Assim, o 80 já se encontra ordenado em sua posição final. Todavia, após a troca, nossa árvore perdeu a propriedade de heap, pois a raiz 13 não é maior que seus filhos, quebrando as regras. Dessa forma, precisamos consertar a heap, fazendo com que a nova raiz “escorregue” até uma posição que restaure nossa árvore binária para ser enquadrada enquanto uma heap. Fazemos isso por meio do método que chamamos de heapifica (em inglês, heapify).

unidade 7

Técnicas de Ordenação - HEAPSORT



unidade 7

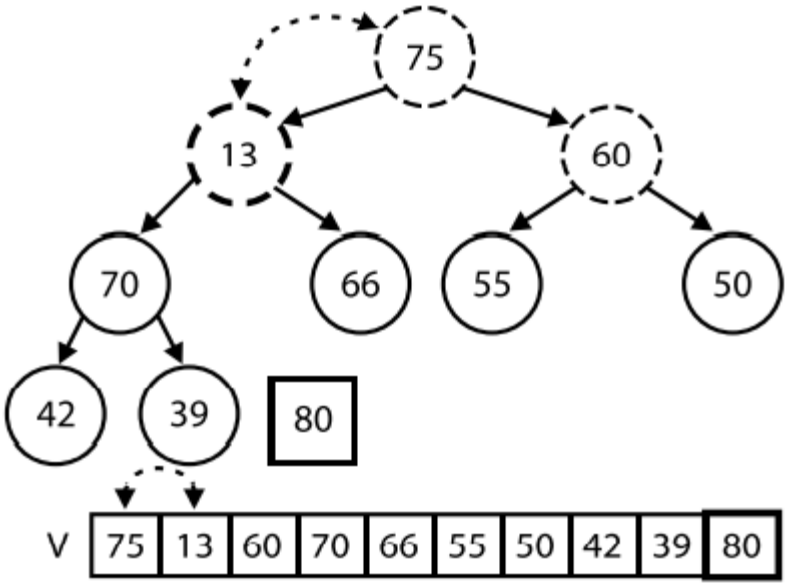
Técnicas de Ordenação - HEAPSORT

O método heapifica, quando invocado, vai comparando um nó pai aos respectivos nós filhos. Caso algum dos filhos seja maior que o nó pai, então realizamos a troca entre o maior filho e o pai, de forma que, após essa operação, o nó pai seja, de fato, maior ou igual aos nós filhos para manter a propriedade da heap. Todavia essa troca pode fazer com que o novo nó filho quebre as propriedades de heap, isto é, o nó filho, recém trocado, pode ter novos filhos que não se categorizam enquanto heap.

unidade 7

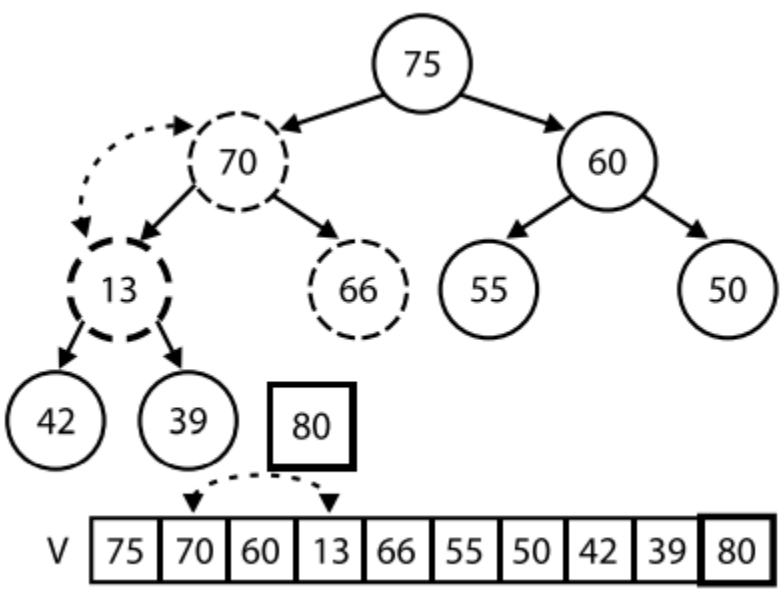
Técnicas de Ordenação - HEAPSORT

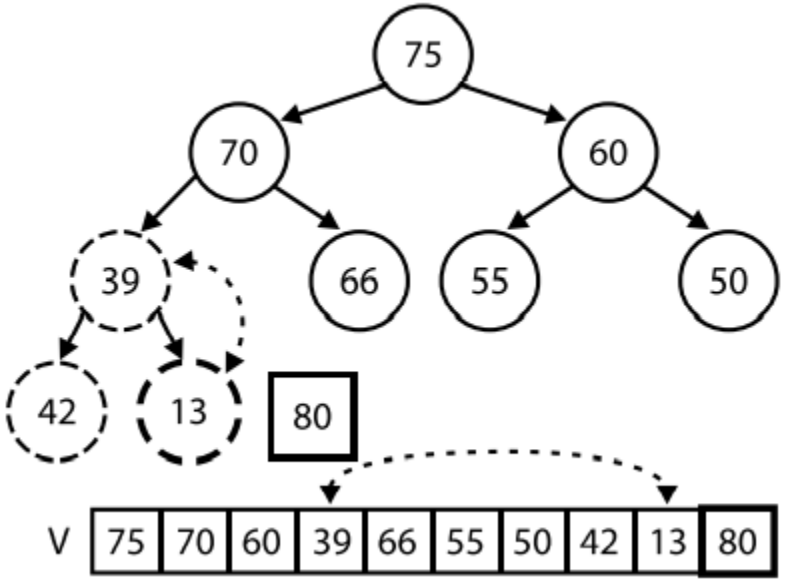
Por isso, é preciso invocar o método heapifica recursivamente, até que todos os nós necessários sejam corrigidos. Observe o passo a passo executado em método heapifica:



unidade 7

Técnicas de Ordenação - HEAPSORT





unidade 7

Técnicas de Ordenação - HEAPSORT

Nas Figuras podemos notar como o método heapifica troca a raiz de uma sub árvore com seu maior filho, à medida em que é executado. Para cada nó trocado, invoca-se o método recursivamente, até que a propriedade de heap seja garantida a todos os nós envolvidos no processo. Além disso, podemos observar que, ao final, temos novamente uma heap na qual o maior entre todos os elementos se encontra na raiz.

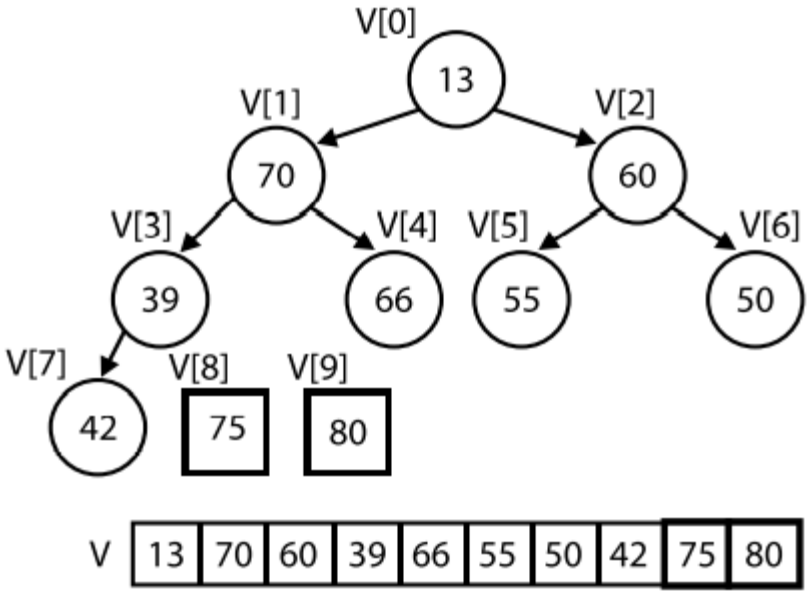
Técnicas de Ordenação - HEAPSORT

Agora, o nó, cuja chave é igual a 75, encontra-se na raiz. Assim, repetimos o processo realizado anteriormente, no qual a raiz da heap era trocada com o “último” elemento do vetor. Por último elemento, entenda a última posição da porção desordenada do vetor, ou seja, a última posição da heap.

unidade 7

Técnicas de Ordenação - HEAPSORT

Nesse caso, vamos trocar o 75 com o 13, fazendo com que o 75 se encaixe em sua posição ordenada final, de acordo com o que podemos visualizar na Figura.



Técnicas de Ordenação - HEAPSORT

A partir daqui, podemos perceber que, repetindo todo o processo descrito até aqui, o Heapsort posiciona os maiores elementos ao final do arranjo, de maneira ordenada. Na Figura anterior foi possível perceber que o valor 75 é o segundo maior entre todos os elementos de v e, corretamente, está alocado à penúltima posição do vetor.

unidade 7

Técnicas de Ordenação - HEAPSORT

```
//Garante as propriedades de heap a um nó
int heapifica(int vec[], int tam, int i){
int e, d, maior, qtd;
qtd = 1;
e = 2*i+1;
d = 2*i+2;
if(e<tam && vec[e] > vec[i]){
maior = e;
}
else {
maior = i;
}
if(d<tam && vec[d] > vec[maior]){
maior = d;
}
if(maior != i){
troca(&vec[i], &vec[maior]);
qtd += heapifica(vec, tam, maior);
}
return qtd;
}
```

```
//Transforma o vetor em uma heap
int constroiHeap(int vec[], int tam){
int i, qtd;
qtd = 0;
for(i=tam/2;i>=0;i--){
qtd += heapifica(vec, tam, i);
}
return qtd;
}
//Ordena com base na estrutura heap
int heapSort(int vec[], int tam){
int n, i, qtd;
qtd = 0;
qtd += constroiHeap(vec, tam);
n = tam;
for(i=tam-1;i>0;i--){
troca(&vec[0], &vec[i]);
n--;
qtd += heapifica(vec, n, 0);
}
return qtd;
}
```

unidade 7

Técnicas de Ordenação - HEAPSORT

Em fim, foi apresentado diversos algoritmos de ordenação. Os processos de pesquisa e ordenação são duas das operações mais utilizadas na computação.

Agora faremos algumas comparações e análise de desempenho dentre seis técnicas.

Para o nosso exercício de raciocínio, vamos definir o esforço computacional. Nessa análise, estaremos considerando como esforço computacional a **quantidade de vezes que o laço mais interno de um algoritmo é repetido ou a quantidade de vezes que uma chamada recursiva é realizada.**

unidade 7

Técnicas de Ordenação - HEAPSORT

Em todas as funções de ordenação descritas foi incluída uma variável chamada **qtd**. Ela será responsável por contar o esforço computacional aplicado na ordenação de cada um dos vetores.

unidade 7

Técnicas de Ordenação - HEAPSORT

A primeira leva de dados será obtido por meio da execução dos seis algoritmos em quatro vetores de dez posições mostrados na Figura.

a) terá uma ordenação totalmente aleatória;

21	48	22	44	40	16	21	10	17	12

b) trará um vetor parcialmente ordenado;

0	1	2	3	4	16	21	10	17	12

c) é um vetor totalmente ordenado;

0	1	2	3	4	5	6	7	8	9

d) um vetor ordenado de forma inversa.

9	8	7	6	5	4	3	2	1	0

unidade 7

Técnicas de Ordenação - HEAPSORT

Na primeira bateria de testes, observou-se que o Bubblesort e o Selectionsort obtiveram o mesmo desempenho. Ambos executam o laço interno até o final, independentemente da situação do vetor em cada passagem. O Shellsort usou metade do esforço que o Insertionsort, o que é muito interessante, pois sabemos que o Shellsort faz várias chamadas do Insertionsort durante a sua execução.

Algoritmo	Esforço Computacional
Bubblesort	45
Selectionsort	45
Mergesort	19
Quicksort	13
Insertionsort	34
Shellsort	15

Técnicas de Ordenação - HEAPSORT

A segunda bateria de testes foi realizada com dados parcialmente ordenados. Tanto o Shellsort com o Insertionsort apresentam desempenho superior aos demais. O Bubblesort e o Selectionsort foram realizados com o mesmo esforço do teste anterior.

Algoritmo	Esforço Computacional
BubbleSort	45
SelectionSort	45
MergeSort	19
QuickSort	19
InsertionSort	6
ShellSort	6

Técnicas de Ordenação - HEAPSORT

O resultado é ainda mais impressionante com os dados totalmente ordenados. Tanto o Insertionsort como o Shellsort fazem apenas uma única passagem pelo vetor de dados sem realizar nenhuma troca. Você deve ter percebido que o Bubblesort e o Selectionsort apresentam sempre o mesmo desempenho, independente da forma como os dados estão armazenados.

Algoritmo	Esforço Computacional
Bubblesort	45
Selectionsort	45
Mergesort	19
Quicksort	21
Insertionsort	0
Shellsort	0

unidade 7

Técnicas de Ordenação - HEAPSORT

O último teste traz o vetor com os dados ordenados de forma decrescente , nesse momento vemos novamente a superioridade do Shellsort em relação ao seu parente próximo, Insertionsort.

Algoritmo	Esforço Computacional
Bubblesort	45
Selectionsort	45
Mergesort	19
Quicksort	19
Insertionsort	45
Shellsort	13

unidade 7

Técnicas de Ordenação - HEAPSORT

O Quicksort, teve um desempenho bem variado, mostrando-se muito eficiente em alguns casos e de desempenho médio em outros.

unidade 7

Técnicas de Ordenação - HEAPSORT

Vamos agora “engrossar um pouco o caldo”. Escolhemos vetores pequenos com tamanho 10 para que possamos reproduzir o teste com os mesmos valores no seu computador. Agora vamos fazer uma nova bateria um pouco mais ousada.

Técnicas de Ordenação - HEAPSORT

A Figura traz uma relação de massas de dados maiores. Usaremos o nosso programa de ambiente de testes para gerar vetores aleatórios nas quantidades descritas e, aplicaremos cada um dos algoritmos e mediremos os seus respectivos esforços.

Teste	Tamanho
Teste a	100
Teste b	1.000
Teste c	10.000

unidade 7

Técnicas de Ordenação - HEAPSORT

Analisando os dados anteriores podemos fazer diversas conclusões. A primeira é que os algoritmos Bubblesort e Selectionsort são os mais lentos e têm o seu consumo computacional exponencial. Quanto maior a massa de dados, maior a quantidade de esforço necessário para a ordenação.

Algoritmo	Esforço Computacional
Bubblesort	4.950
Selectionsort	4.950
Mergesort	199
Quicksort	141
Insertionsort	2.261
Shellsort	328

unidade 7

Técnicas de Ordenação - HEAPSORT

O Shellsort mostra desempenho superior ao do Insertionsort mesmo que o primeiro faça inúmeras chamadas ao segundo durante o seu funcionamento.

Algoritmo	Esforço Computacional
Bubblesort	499.500
Selectionsort	499.500
Mergesort	1.999
Quicksort	1.903
Insertionsort	241.493
Shellsort	5.959

unidade 7

Técnicas de Ordenação - HEAPSORT

O Quicksort, que se mostrou mediano nos vetores de dez posições, obteve o melhor desempenho em todos os testes com grandes quantidades de dados. O Mergesort, que também utiliza o conceito de dividir para conquistar, não é tão rápido quanto o Quicksort, mas tem desempenho superior aos outros métodos de ordenação aqui apresentados.

Algoritmo	Esforço Computacional
Bubblesort	49.995.000
Selectionsort	49.995.000
Mergesort	19.999
Quicksort	19.903
Insertionsort	24.722.325
Shellsort	71.682

unidade 7

Técnicas de Ordenação - HEAPSORT

Quando é preciso ordenar um pequeno volume de dados, você pode se dar ao luxo de fazer rapidamente implementações mais simples, porque o tempo de execução é tão pequeno que compensa o tempo economizado na hora de codificar um método mais complexo.

Já para o caso de estarmos ordenando índices de arquivos grandes, como o usado no sistema operacional para localizar arquivos, ou por bancos de dados para aperfeiçoar a busca em suas tabelas, a implementação de algoritmos mais rápidos se torna crucial.

unidade 7

Referências:

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. Estruturas de dados e seus algoritmos. 2ed. Rio de Janeiro: LTC, 1994. 320p.

TENENBAUM, Aaron M.; LANGSAM, Yedidyah; AUGENSTEIN, Moshé J.. Estruturas de dados usando C. São Paulo: Makron Books, 1995. 884p.

VELOSO, Paulo et al.. Estruturas de dados. Rio de Janeiro: Campus, 2001. 228p

Atividades - unidade 7


Unicesumar