## Análise de Algoritmos

## Alunos:

## Jean Carlos Rodrigues RA 1312588

## Elias Scheifer RA 1291840

## Métodos de ordenação

## QuickSort

O [algoritmo](http://pt.wikipedia.org/wiki/Algoritmo) **Quicksort** é um método de ordenação muito rápido e eficiente, inventado por [C.A.R. Hoare](http://pt.wikipedia.org/wiki/C.A.R._Hoare) em 1960, quando visitou a [Universidade de Moscovo](http://pt.wikipedia.org/wiki/Universidade_de_Moscovo) como estudante. Naquela época, Hoare trabalhou em um projeto de [tradução de máquina](http://pt.wikipedia.org/w/index.php?title=Tradu%C3%A7%C3%A3o_de_m%C3%A1quina&action=edit&redlink=1) para o [Na Physical Laboratory tional](http://pt.wikipedia.org/w/index.php?title=National_Physical_Laboratory,_UK&action=edit&redlink=1) . Ele criou o *Quicksort* ao tentar traduzir um dicionário de inglês para russo, ordenando as palavras, tendo como objetivo reduzir o problema original em subproblemas que possam ser resolvidos mais facil e rapidamente.

A idéia básica é dividir o problema de ordenar um conjunto com n itens em dois sub-problemas menores. Os problemas menores são ordenados independentemente, os resultados são combinados para produzir a solução final.

Este método baseia-se na divisão da tabela em duas sub-tabelas, dependendo de um elemento chamado pivô, normalmente o 1º elemento da tabela. Uma das sub-tabelas contém os elementos menores que o pivô enquanto a outra contém os maiores. O pivô é colocado entre ambas, ficando na posição correcta. As duas sub-tabelas são ordenadas de forma idêntica, até que se chegue à tabela com um só elemento.

 Este método baseia-se na divisão da tabela em duas sub-tabelas, dependendo de um elemento chamado pivô, normalmente o 1º elemento da tabela. Uma das sub-tabelas contém os elementos menores que o pivô enquanto a outra contém os maiores. O pivô é colocado entre ambas, ficando na posição correcta. As duas sub-tabelas são ordenadas de forma idêntica, até que se chegue à tabela com um só elemento.

Complexidade: caso médio O(N \* log N).Esse método de ordenação divide-se em vários passos:

Escolher para pivô o primeiro elemento da tabela (p=x[1])

Se os elementos de x forem rearranjados de forma a que o pivô (p) sejam colocados na posição j e sejam respeitadas as seguintes condições:

1. todos os elementos entre as posições 1 e j-1 são menores ou iguais que o pivô (p)

2. todos os elementos entre as posições j+1 e n são maiores que o pivô (p) Então p permanecerá na posição j no final do ordenamento.

Se este processo for repetido para as sub-tabelas x[1] a x[j-1] e x[j+1] a x[n] e para todas as sub-tabelas criadas nas iterações seguintes obteremos no final uma tabela ordenada.

Portanto a parte mais difícil deste método é o procedimento parte que divide a tabela em 2 sub-tabelas dependendo do pivô.

## Merge sort

O ***merge sort***, ou [ordenação](http://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_(computa%C3%A7%C3%A3o)) por mistura, é um exemplo de [algoritmo de ordenação](http://pt.wikipedia.org/wiki/Algoritmo_de_ordena%C3%A7%C3%A3o) do tipo dividir para conquistar

Sua ideia básica consiste em Dividir(o problema em vários sub-problemas e resolver esses sub-problemas através da recursividade) e Conquistar(após todos os sub-problemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos sub-problemas).Como o algoritmo do Merge Sort usa a recursividade em alguns problemas esta técnica não é muito eficiente devido ao alto consumo de memória e tempo de execução.

Baseia-se em junções sucessivas (merge) de 2 seqüências ordenadas em uma única seqüência ordenada . Aplica um método “dividir para conquistar” divide o vetor em 2 segmentos (sub-vetores) de comprimento n/2 e n/2 ordena recursivamente cada subvetor (dividindo novamente, quando possível) faz o merge dos 2 sub-vetores ordenados para obter o vetor ordenado completo.

## Shell sort

**Shell sort** (também conhecido como uma extensão de insert sort) é o mais eficiente algoritmo de classificação dentre os de complexidade quadrática. É um refinamento do método de inserção direta.O algoritmo difere do método de inserção direta pelo fato de no lugar de considerar o array a ser ordenado como um único segmento, ele considera vários segmentos sendo aplicado o método de inserção direta em cada um deles. Basicamente o algoritmo passa várias vezes pela lista dividindo o grupo maior em menores. Nos grupos menores é aplicado o método da ordenação por inserção.

Problema com o algoritmo de ordenação por inserção.Troca itens adjacentes para determinar o ponto de inserção.São efetuadas n - 1 comparações e movimentações quando o menor item está na posição mais à direita no vetor. O método de Shell contorna este problema permitindo trocas de registros distantes um do outro.

## Algoritmos

Quick Sort:

void swap(int\* a, int\* b) {

int tmp;

tmp = \*a;

\*a = \*b;

\*b = tmp;

}

int partition(int vec[], int left, int right) {

int i, j;

i = left;

for (j = left + 1; j <= right; ++j) {

if (vec[j] < vec[left]) {

++i;

swap(&vec[i], &vec[j]);

}

}

swap(&vec[left], &vec[i]);

return i;

}

void quickSort(int vec[], int left, int right) {

int r;

if (right > left) {

r = partition(vec, left, right);

quickSort(vec, left, r - 1);

quickSort(vec, r + 1, right);

}

}

MergeSort:

void merge(int vec[], int vecSize) {

int mid;

int i, j, k;

int\* tmp;

tmp = (int\*) malloc(vecSize \* sizeof(int));

if (tmp == NULL) {

exit(1);

}

mid = vecSize / 2;

i = 0;

j = mid;

k = 0;

while (i < mid && j < vecSize) {

if (vec[i] < vec[j]) {

tmp[k] = vec[i++];

}

else {

tmp[k] = vec[j++];

}

++k;

}

if (i == mid) {

while (j < vecSize) {

tmp[k++] = vec[j++];

}

}

else {

while (i < mid) {

tmp[k++] = vec[i++];

}

}

for (i = 0; i < vecSize; ++i) {

vec[i] = tmp[i];

}

free(tmp);

}

void mergeSort(int vec[], int vecSize) {

int mid;

if (vecSize > 1) {

mid = vecSize / 2;

mergeSort(vec, mid);

mergeSort(vec + mid, vecSize - mid);

merge(vec, vecSize);

}

}

ShellSort:

void shellSort(int \*vet, int size) {

int i , j , value;

int gap = 1;

do {

gap = 3\*gap+1;

} while(gap < size);

do {

gap /= 3;

for(i = gap; i < size; i++) {

value = vet[i];

j = i - gap;

while (j >= 0 && value < vet[j]) {

vet [j + gap] = vet[j];

j -= gap;

}

vet [j + gap] = value;

}

} while ( gap > 1);

}

## Análise de Tempo dos Algoritmos

Os algoritmos MergeSort, QuickSort, e ShellSort são mais rápidos por isso precisam ser analisados em separado para visualização dos tempos, para um grande número de elementos o QuickSort se mostrou até 50% mais rápido que o MergeSort, e 100 % mais rápido que o MergeSort.

Os algoritmos MergeSort, QuickSort e ShellSort tem um tempo quase linear, dos algoritmos análisados o BubbleSort se revelou mais lento que os demais.