Introdução

Nós implementamos e realizamos uma séries de testes com os principais algoritmos de ordenação existentes: Bubble Sort, Insertion Sort, Merge Sort, Quick Sort e Heap Sort. Estes foram implementados para ordenar um vetor de números inteiros, tendo como entrada de testes 10 vetores para o melhor caso e 10 vetores pra pior caso de cada algoritmo, além de mais cinquenta vetores aleatórios e de tamanho variável que serão usados para todos eles.

Algoritmos

Os algoritmos originais foram implementados em C e o tempo foi medido em microssegundos usando o método *gettimeofday()* da biblioteca *sys/time.h*, tirando a diferença do tempo final e inicial de cada execução do algoritmo.

Abaixo o **pseudo-código** das linguagens implementadas:

Bubble Sort:

```
Bubble(V[], n)
    k = n - 1
    para (i = 0; enquanto i < n ) faça
        para (j = 0; enquanto j < k) faça
            se (v[j] > v[j + 1]) faça
                aux = v[j]
                v[j] = v[j + 1]
                v[j + 1] = aux
            j++
        k--
        i++
```

Insertion Sort:

```
Insertion(V[], n)
    para (i = 1 ; enquanto i < n)
        enquanto ((i != 0) e (v[i - 1] > v[i]))
            aux = v[i]
            v[i] = v[i - 1]
            v[i - 1] = aux
            i--
        i++
```

Merge Sort:

```
Merge(V[], inicio, fim)
    se inicio = fim
        retorne

    meio = (inicio + meio)/2
    Merge(V, inicio, meio)
    Merge(V, meio + 1, fim)
    Merge2(V, inicio, meio, fim)
```

```
Merge2(v[], inicio, meio, fim)

    i=inicio
    j=meio+1
    k=0
    enquanto(i<meio+1 ou j< fim+1) faça
        se(i=meio+1) faça
            vetorTemp[k]=v[j]
            j++
            k++
        senão
            se( j= fim+1) faça
                vetorTemp[k]=v[i]
                i++
                k++
            senão
                se(v[i]<v[j])
                    vetorTemp[k]=v[i]
                    i++
                    k++
                senão
                    vetorTemp[k]=v[j]
                    j++
                    k++

    para(i=inicio ; enquanto i≤ fim) faça
        v[i]=vetorTemp[i−inicio]
        i=i++
```

Quick Sort:

```
Quick(a, l, r)
    se (l < r) faça
        j = particiona(a, l, r)
        quick(a, l, j − 1)
        quick(a, j + 1, r)
```

```
particiona(a, l, r)
    pivot = a[l]
    i = l
    j = r + 1

    enquanto (1)
        faça
            ++i
        enquanto (a[i] ≤ pivot ∧¿ i ≤ r)
        faça
            −−j
        enquanto (a[j] > pivot)
        se (i ≥ j)
            encerre laço
        t = a[i]
        a[i] = a[j]
        a[j] = t

    t = a[l]
    a[l] = a[j]
    a[j] = t
    retorne j
```

Heap Sort:

```
Heap(v, n)
    i = n / 2
    loop infinito
        se (i > 0) faça
            i−−
            t = v[i]
        senão
            n−−
            se (n = 0) faça
                retorne
            t = v[n]
            v[n] = v[0]
        pai = i
        filho = i * 2
        enquanto (filho < n)
            se ((filho + 1 < n) e (v[filho + 1] > v[filho])) faça
                filho++
            se (v[filho] > t) faça
                v[pai] = v[filho]
                pai = filho
                filho = pai * 2 + 1
            senão
                encerra laço
        v[pai] = t
```

Dados e Execução

Os arquivos-fontes "*.cpp*" estão na mesma pasta dos arquivos contendo os vetores de melhores casos, na raiz do projeto (*Bubble_M.txt, Insertion_M.txt, Merge_M.txt, Quick_M.txt, Heap_M.txt)* junto com os de piores casos (*Bubble_P.txt, Insertion_P.txt, Merge_P.txt, Quick_P.txt, Heap_P.txt*) e os gerados aleatoriamente (Aleatorios.txt).

Piores casos:

- Bubble, Insertion, Quick: vetor na ordem decrescente;
- Merge: vetor ordenado com duas metades intercaladas entre si, pois ai ele teria que particionar todos;
- Heap: vetor ordenado.

Melhor casos:

- Bubble, Insertion, Merge: vetor ordenado;
- Quick: o melhor caso é quando as partições tem sempre o mesmo tamanho (n/2), porém ele tem eficiência igual a do caso médio, pois na prática a diferença é muito pequena, ambos O(nlogn). Então consideraremos como melhor caso um vetor aleatório que não é o pior caso;
- Heap: vetor ordenado em ordem decrescente, por ser um heapsort usando shiftdown.

Testes e Resultados

Os testes foram realizados num laço na *main.cpp* passando como parâmetro o algoritmo e o arquivo com os vetores em questão. O projeto se encontra num repositório do GitHub e é aberto:

https://github.com/douglasascosta/mc458.

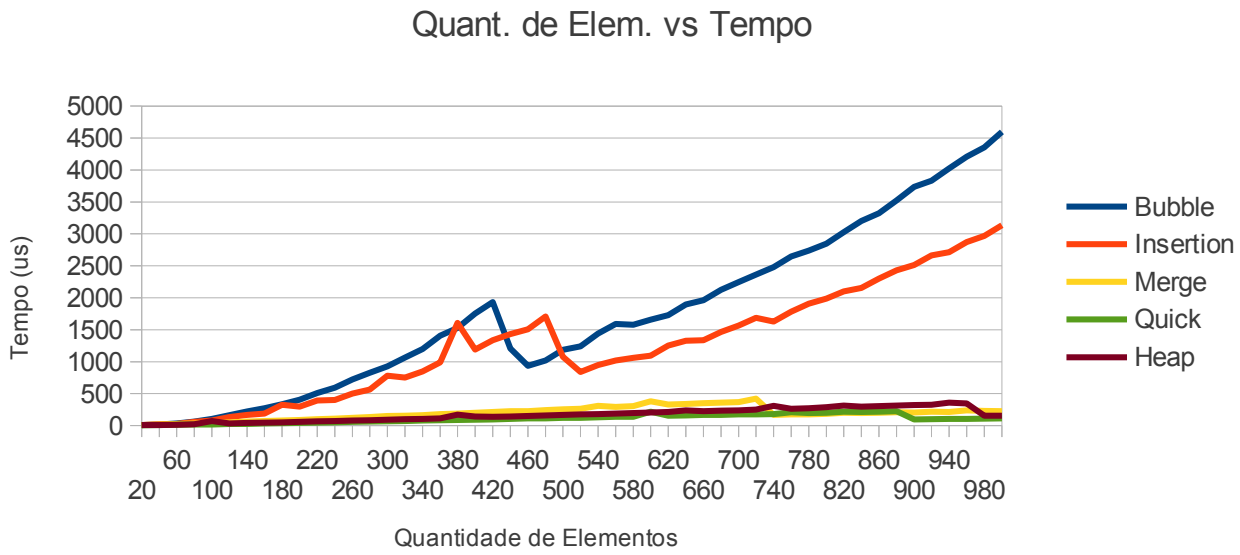
Os dados foram obtidos de uma máquina com processador Intel Core i5 M460 2.53 MHz x4, 4GB de memória RAM, no SO Ubuntu 13.04 x64.

Após 100 medidas de **melhor e pior caso**, nós calculamos as médias de tempo para cada algoritmo em (tempo em µs):

	Bubble		Insertion		Merge		Quick		Heap	
	M	P	M	P	M	P	M	P	M	P
Médias (µs)	25,05	49,09	1,49	71,02	17,69	18,42	10,9	23,47	11,91	13,63

Teste de **vetores aleatórios** para todos os algoritmos (tempo em μs):

n	**Bubble**	**Insertion**	**Merge**	**Quick**	**Heap**
20	6	8	9	5	6
40	17	20	15	8	9
60	35	27	24	11	14
80	63	57	32	15	20
100	104	78	41	17	71
120	162	133	50	24	32
140	220	166	61	27	40
160	269	188	70	32	45
180	332	327	80	37	52
200	403	297	88	42	57
220	508	392	99	47	65
240	591	401	110	51	71
260	722	499	123	56	78
280	827	564	134	62	84
300	928	779	148	67	91
320	1066	750	155	71	99
340	1198	845	162	77	106
360	1408	989	179	83	113
380	1528	1606	187	87	171
400	1754	1191	200	90	142
420	1932	1334	212	95	139
440	1207	1433	224	104	143
460	934	1508	227	112	151
480	1020	1708	242	112	158
500	1186	1078	255	119	167
520	1241	838	264	121	174
540	1442	949	308	127	181
560	1590	1019	293	135	188
580	1578	1061	306	138	195
600	1658	1094	379	216	203
620	1729	1253	328	152	211
640	1894	1326	338	159	238
660	1962	1337	350	165	223
680	2124	1465	358	165	233
700	2245	1566	367	173	239
720	2361	1687	420	176	251
740	2480	1629	169	181	310
760	2645	1782	173	208	265
780	2739	1908	176	197	272
800	2846	1989	183	200	286
820	3026	2099	202	215	314
840	3203	2153	195	213	295
860	3321	2299	200	217	306
880	3521	2429	210	225	312
900	3736	2513	204	97	321
920	3831	2664	217	100	327
940	4023	2713	213	103	358
960	4208	2871	238	106	345
980	4355	2967	229	109	152
1000	4596	3137	225	113	155



Fontes bibliográficas

U. Manber, Algorithms: A Creative Approach, Addison-Wesley.

T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, MIT Press.

http://linux.die.net/man/2/gettimeofday

http://www.ft.unicamp.br/liag/siteEd/includes/arquivos/MergeSortResumo_Grupo4_ST364A_2010.pdf

http://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/quicksort.pdf