Projeto e análise de algoritmos

João Pedro Peres Bertoncelo

RA112650

(Comentários serão feitos na análise do tempo de execução dos algoritmos)

1- Inserção Direta:

```
void insercao_direta(int vetor[], int tamanho){
        int i, j, chave, trocas;
  srand(time(0));
  for(i=0;i<tamanho;i++)
    {
    vetor[i]=rand()%100;
    printf("Vetor [%2d]: %3d\n",i+1,vetor[i]);
    }
  for (j=1;j<tamanho;j++)
    {
    chave = vetor[j];
    i = j - 1;
    while((i>=0) && (vetor[i]>chave))
      {
      vetor[i+1] = vetor[i];
      i = i - 1;
```

```
        trocas++;
        }
      vetor[i+1] = chave;
      }
}
```

Tempo de Execução(Físico/Real):

Aleatório:3,364 seg – Caso médio de execução, será O(n^2)

Crescente:2,810 seg – Melhor caso, será O(n)

Decrescente:2,822 seg – Pior caso, será O(n^2)

2-BubbleSort

```
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 - k; j++) {
         printf("%d, ", j);

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

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

Tempo de Execução(Físico/Real):

Aleatório:3,572 seg – Teóricamente deveria ser o caso médio, mas em testes pessoais acabou sendo o melhor caso, e é O(n^2)

Crescente:7,174 seg – Em meus testes pessoais acabou sendo o pior caso, apesar de normalmente ser o melhor, e ser O(n)

Decrescente:4,239 seg – Em meus testes acabou sendo o caso médio, mas deveria ser o pior caso, sendo O(n^2)

3-Seleção Direta

```
void selecao_direta(int vetor[], int tamanho)
{
  int i, j, menor, aux;

  for (i = 0; i < tamanho - 1; ++i)
  {
    menor = i;
    for (j = i + 1; j < tamanho; ++j)
    {
      if (vetor[j] < vetor[menor])
```

```
      menor = j;
   }
   aux = vetor[i];
   vetor[i] = vetor[menor];
   vetor[menor] = aux;
  }
}
```

Tempo de Execução(Físico/Real):

Aleatório:1,865 seg – assim como esperado, foi o caso médio, sendo O(n^2)

Crescente:6,246 seg – Deveria ser o melhor caso, mas no meu pessoal foi o pior caso, sendo O(n^2)

Decrescente:1,755 seg – Deveria ser o pior caso, porem nos meus testes foi o melhor, sendo O(n^2)

4-Shellsort

```
void shellSort(int *vet, int size) {
   int i, j, value;

   int h = 1;
   while(h < size) {
      h = 3*h+1;
   }
   while (h > 0) {
```

```
      for(i = h; i < size; i++) {
         value = vet[i];
         j = i;
         while (j > h-1 && value <= vet[j - h]) {
            vet[j] = vet[j - h];
            j = j - h;
         }
         vet[j] = value;
      }
      h = h/3;
   }
}
```

Tempo de Execução(Físico/Real):

Aleatório:2,874 seg – Assim como o esperado, foi o caso médio, sendo O(n log n)

Crescente:2,190 seg – Assim como o esperado, foi o melhor caso, sendo O(n log n)

Decrescente:4,009 seg – Assim como o esperado, foi o pior caso, dendo O(n log n)

5-QuickSort

```
int
separa (int v[], int p, int r) {
  int c = v[r];
```

```
  int t, j = p;
  for (int k = p; k < r; ++k)
    if (v[k] <= c) {
      t = v[j], v[j] = v[k], v[k] = t;
      ++j;
    }
  t = v[j], v[j] = v[r], v[r] = t;
  return j;
}

void quickSort (int vetor[], int p, int r)
{
  while (p < r) {
    int j = separa (vetor, p, r);
    if (j - p < r - j) {
      quickSort (vetor, p, j-1);
      p = j + 1;
    } else {
      quickSort (vetor, j+1, r);
      r = j - 1;
    }
  }
}
```

Tempo de Execução(Físico/Real):

Aleatório:2,581 seg

Crescente:2,636 seg

Decrescente:2,527 seg

Caso sejam partições balanceadas: O(n log n)

Caso sejam partições desbalanceadas: O(n^2)

6-HeapSort

```
void peneira(int *vet, int raiz, int fundo) {
	int pronto, filhoMax, tmp;

	pronto = 0;
	while ((raiz*2 <= fundo) && (!pronto)) {
		if (raiz*2 == fundo) {
			filhoMax = raiz * 2;
		}
		else if (vet[raiz * 2] > vet[raiz * 2 + 1]) {
			filhoMax = raiz * 2;
		}
		else {
			filhoMax = raiz * 2 + 1;
		}

	if (vet[raiz] < vet[filhoMax]) {
```

```
                tmp = vet[raiz];
                vet[raiz] = vet[filhoMax];
                vet[filhoMax] = tmp;
                raiz = filhoMax;
    }
        else {
     pronto = 1;
        }
  }
}

void heapsort(int vetor[], int n) {
        int i, tmp;

        for (i = (n / 2); i >= 0; i--) {
                peneira(vetor, i, n - 1);
        }

        for (i = n-1; i >= 1; i--) {
                tmp = vetor[0];
                vetor[0] = vetor[i];
                vetor[i] = tmp;
                peneira(vetor, 0, i-1);
        }
```

```
}
```

Tempo de Execução(Físico/Real):

Aleatório:1,601 seg

Crescente:2,971 seg

Decrescente:1,434 seg

Em todos os casos será O(n log n)

7-MergeSort

```
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);
}

void mergeSort(int vetor[], int comeco, int fim){
   if (comeco < fim) {
      int meio = (fim+comeco)/2;
```

```
        mergeSort(vetor, comeco, meio);
        mergeSort(vetor, meio+1, fim);
        merge(vetor, comeco, meio, fim);
    }
}
```

Tempo de Execução(Físico/Real):

Aleatório:0,8371 seg

Crescente:1,861 seg

Decrescente:1,563 seg

Em todos os casos, será O(n log n)

8-Radixsort

```
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);
}
```

Tempo de Execução(Físico/Real):

Aleatório:1,677 seg

Crescente:3,179 seg

Decrescente:1,901 seg

Em todos os casos, será $\Theta(d(n + k))$