MC3305
Algoritmos e Estruturas de Dados II

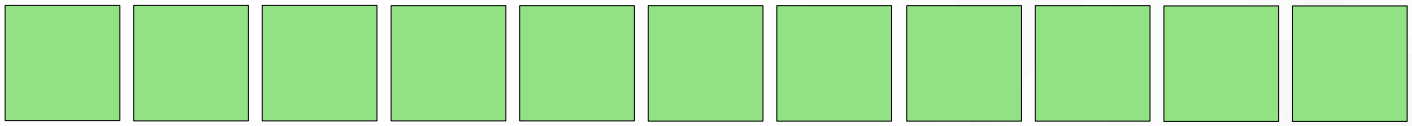
Aula 05 – Ordenação parcial

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

2Q-2015

Ordenação parcial (seleção do k-éssimo maior)

- Consiste em obter os ***k*** primeiros elementos de um vetor ordenado com ***n*** elementos.
- Quando ***k=1*** o problema se reduz a encontrar o mínimo (ou o máximo) de um conjunto de elementos.
- Quando ***k=n*** caímos no problema clássico de ordenação.



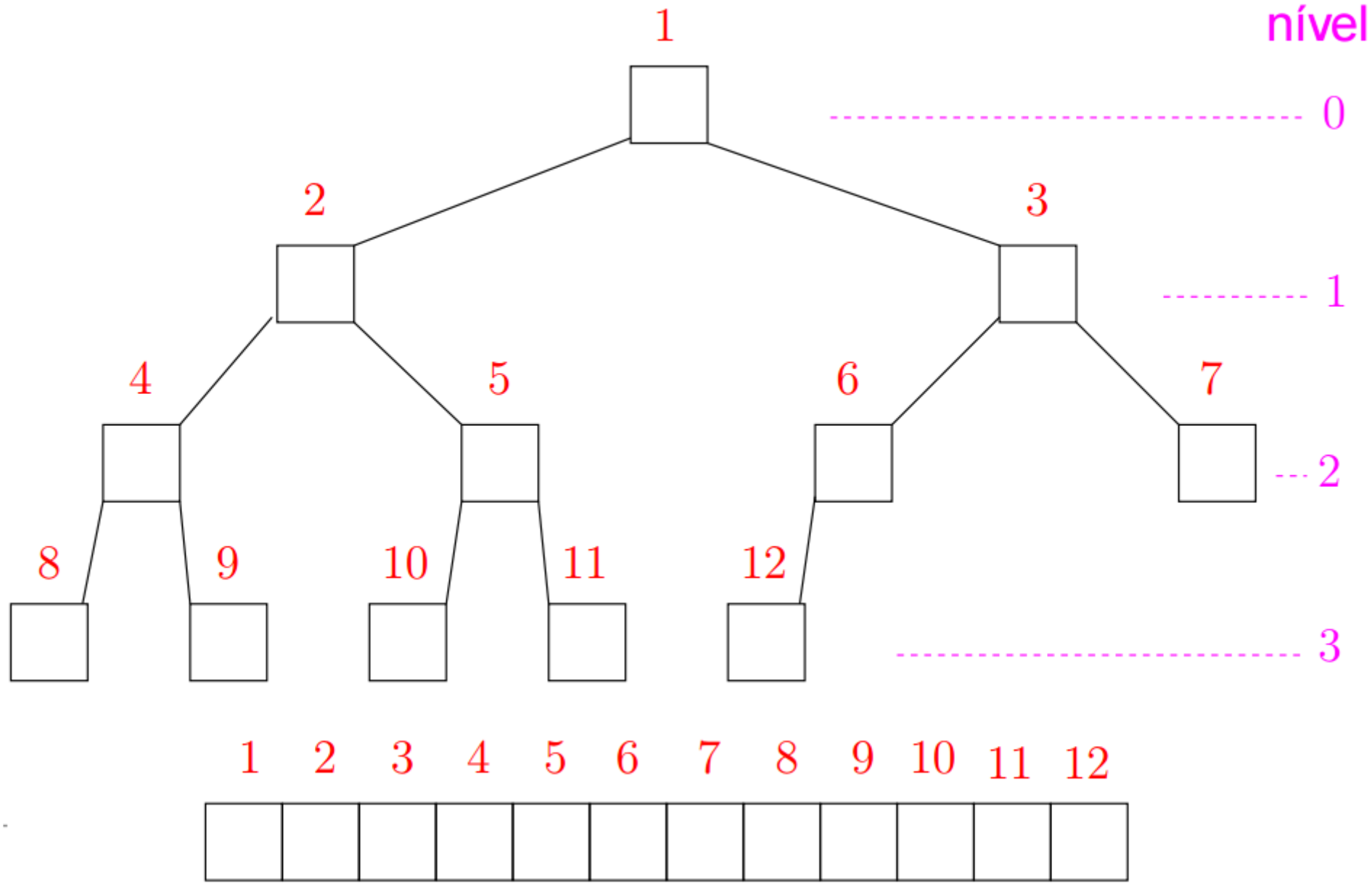
Ordenação parcial

Os algoritmos de Ord. Parcial que estudaremos serão:

- Seleção parcial
- Inserção parcial
- Heapsort parcial
- Quicksort parcial (←lista 01)

Representação de árvores em vetores

Slides baseados no material de aula do Prof. José Coelho de Pina (IME/USP)


1
nível
0
2
3
1
4
5
6
7
2
8
9
10
11
12
3
1 2 3 4 5 6 7 8 9 10 11 12

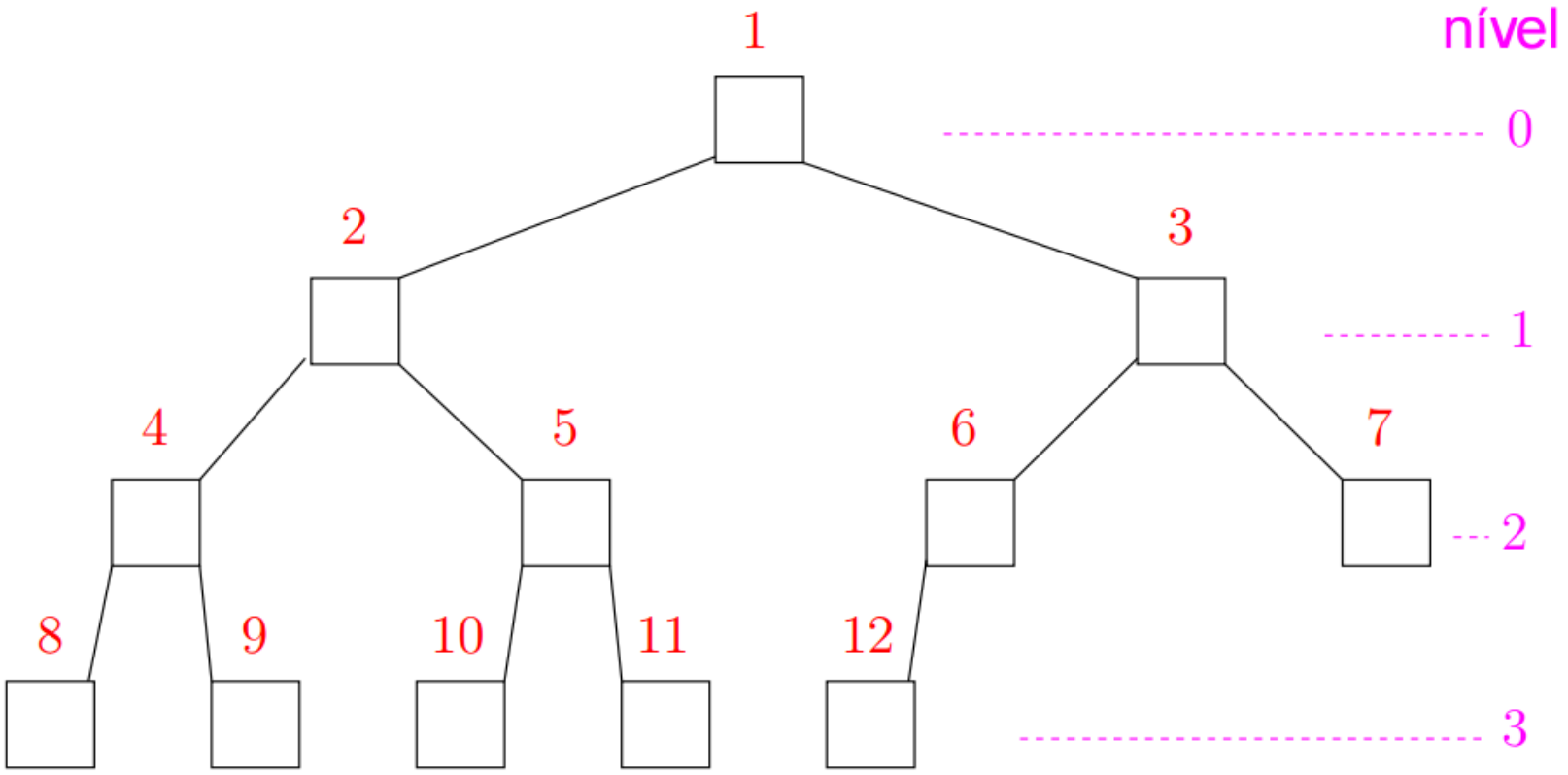
Pais e filhos

- `A[1..m]` é um vetor representado por uma árvore
- Diremos que para qualquer índice ou nó `i`:
  - `floor(i/2)` é o pai de `i`.
  - `2*i` é o filho esquerdo de `i`.
  - `2*i+1` é o filho direito de `i`.
- O nó `1` não tem pai e é chamado de nó raiz.
- Um nó `i` só tem filho esquerdo se $2*i \leq m$.
- Um nó `i` só tem filho direito se $2*i+1 \leq m$.
- Um no `i` é um nó folha, se não tem filhos: $2*i > m$.

Pais e filhos

- Todo nó `i` é raiz da sub-árvore formada por:

$$A[i, 2i, 2i+1, 4i, 4i+1, 4i+2, 4i+3, 8i, \ldots, 8i+7, \ldots]$$



Propriedades

`A[1..m]` é um vetor representado por uma árvore

filho esquerdo de i:	$2\,i$
filho direito de i:	$2\,i + 1$
pai de i:	$\lfloor i/2 \rfloor$
nível da raiz:	0
nível de i:	$\lfloor \lg i \rfloor$
altura da raiz:	$\lfloor \lg m \rfloor$
altura da árvore:	$\lfloor \lg m \rfloor$
altura de i:	$\lfloor \lg(m/i) \rfloor$
altura de uma folha:	0
total de nós de altura h	$\leq \lceil m/2^{h+1} \rceil$

Heapsort parcial

Heapsort parcial

Utiliza um tipo abstrato de dados min-heap para informar o menor item do conjunto.

Usando um MIN-HEAP

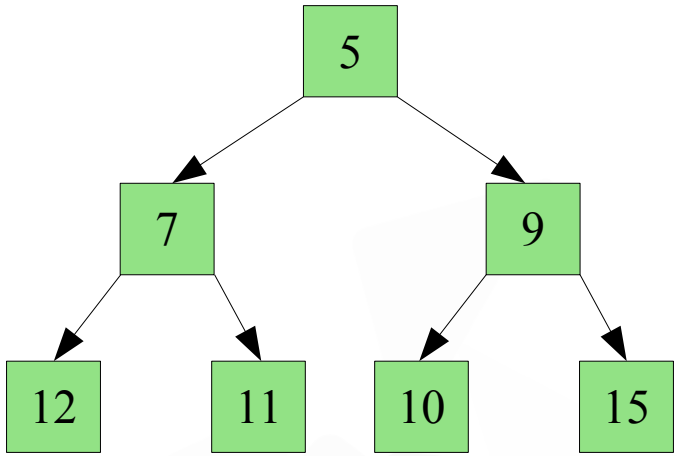
*Na primeira iteração, o **menor** item que está em V[1] (raiz do heap) é trocado com o item que está em V[n-1].*

Em seguida o heap é refeito.

*Novamente o **menor** está em V[1], troque-o com V[n-2].*

Repita as duas últimas operações até que o k-ésimo menor esteja seja trocado com V[n-k].

Ao final, os k menores estão nas k últimas posições do vetor V.



Animação: https://www.cs.usfca.edu/~galles/visualization/HeapSort.html

Heapsort parcial

Utiliza um tipo abstrato de dados min-heap para informar o menor item do conjunto.

Usando um MIN-HEAP

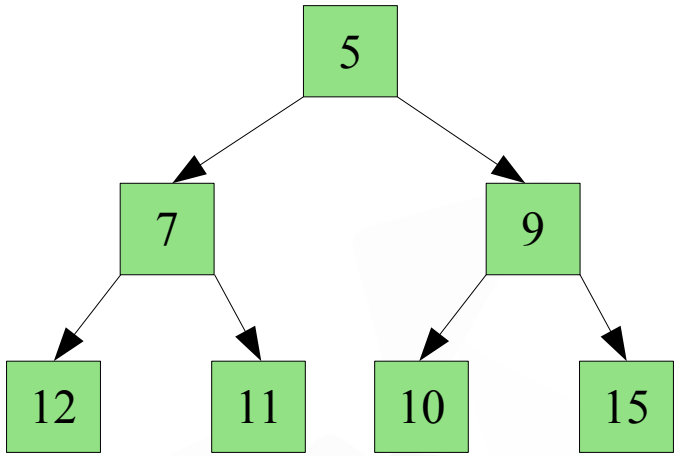
*Na primeira iteração, o **menor** item que está em V[1] (raiz do heap) é trocado com o item que está em V[n-1].*

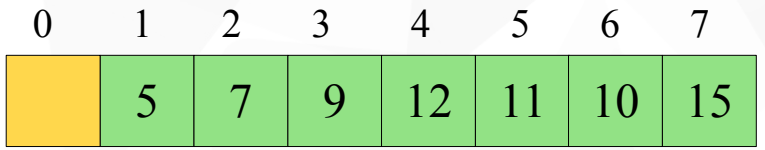
Em seguida o heap é refeito.

*Novamente o **menor** está em V[1], troque-o com V[n-2].*

Repita as duas últimas operações até que o k-ésimo menor esteja seja trocado com V[n-k].

Ao final, os k menores estão nas k últimas posições do vetor V.





Animação: https://www.cs.usfca.edu/~galles/visualization/HeapSort.html

Heapsort parcial

```
void MinHeapify (int A[], int m, int i) {
    int e, d, menor, aux;
    e = 2*i;
    d = 2*i+1;
    if (e<=m && A[e]<A[i])
        menor = e;
    else
        menor = i;
    if (d<=m && A[d]<A[menor])
        menor = d;
    if (menor!=i) {
        aux = A[menor];
        A[menor] = A[i];
        A[i] = aux;
        MinHeapify(A, m, menor);
    }
}
```

```
void HeapSort(int A[], int n) {
    int i, m, aux;
    BuildMinHeap(A, n);
    m = n;
    for (i=n; i>=1; i--) {
        aux = A[i];
        A[i] = A[1];
        A[1] = aux;
        m = m-1;
        MinHeapify (A, m, 1);
    }
}
```

```
void BuildMinHeap(int A[], int n) {
    int i;
    for (i=n/2; i>=1; i--)
        MinHeapify (A, n, i);
}
```

Heapsort parcial

```
void MinHeapify (int A[], int m, int i) {
    int e, d, menor, aux;
    e = 2*i;
    d = 2*i+1;
    if (e<=m && A[e]<A[i])
        menor = e;
    else
        menor = i;
    if (d<=m && A[d]<A[menor])
        menor = d;
    if (menor!=i) {
        aux = A[menor];
        A[menor] = A[i];
        A[i] = aux;
        MinHeapify(A, m, menor);
    }
}
```

```
void BuildMinHeap(int A[], int n) {
    int i;
    for (i=n/2; i>=1; i--)
        MinHeapify (A, n, i);
}
```

```
void HeapSort(int A[], int n) {
    int i, m, aux;
    BuildMinHeap(A, n);
    m = n;
    for (i=n; i>=1; i--) {
        aux = A[i];
        A[i] = A[1];
        A[1] = aux;
        m = m-1;
        MinHeapify (A, m, 1);
    }
}
```

```
void PartialHeapSort(int A[], int n, int k) {
    int i, m, aux;
    BuildMinHeap(A, n);
    m = n;
    for (i=n; i>n-k; i--) {
        aux = A[i];
        A[i] = A[1];
        A[1] = aux;
        m = m-1;
        MinHeapify (A, m, 1);
    }
}
```

Heapsort parcial

- O **BuildMinHeap** deve construir o heap a um custo O(n).
- O prodecimento **MinHeapify** (arruma o heap) tem um custo de O(lg(n)).
- O procedimento **PartialHeapSort** chama o procedimento **MinHeapify** k vezes.
- Complexidade:

$$O(n + k \log n)$$

Heapsort parcial

- O **BuildMinHeap** deve construir o heap a um custo O(n).
- O prodecimento **MinHeapify** (arruma o heap) tem um custo de O(lg(n)).
- O procedimento **PartialHeapSort** chama o procedimento **MinHeapify** k vezes.
- Complexidade:

$$O(n + k \log n) = \begin{cases} O(n) & \text{se } k \leq \frac{n}{\log n} \\ O(k \log n) & \text{se } k > \frac{n}{\log n} \end{cases}$$

Heapsort parcial

```
int main()
{
    int v[] = {999,15,14,13,12,11,10,-1,0,1,2,3,4,5,6,7,8,9};
    int n=sizeof(v)/sizeof(v[0]);

    ImprimeVetor(v, n);
    PartialHeapSort(v, n-1, 7);
    ImprimeVetor(v, n);
}
```

```
15 14 13 12 11 10 -1 0 1 2 3 4 5 6 7 8 9
```

```
6 7 10 8 11 12 15 9 14 13 5 4 3 2 1 0 -1
```

Comparação empírica dos algoritmos

n, k	Seleção	Quicksort	Inserção	Inserção2	Heapsort
$n: 10^1 \quad k: 10^0$	1	2,5	1	1,2	1,7
$n: 10^1 \quad k: 10^1$	1,2	2,8	1	1,1	2,8
$n: 10^2 \quad k: 10^0$	1	3	1,1	1,4	4,5
$n: 10^2 \quad k: 10^1$	1,9	2,4	1	1,2	3
$n: 10^2 \quad k: 10^2$	3	1,7	1	1,1	2,3
$n: 10^3 \quad k: 10^0$	1	3,7	1,4	1,6	9,1
$n: 10^3 \quad k: 10^1$	4,6	2,9	1	1,2	6,4
$n: 10^3 \quad k: 10^2$	11,2	1,3	1	1,4	1,9
$n: 10^3 \quad k: 10^3$	15,1	1	3,9	4,2	1,6
$n: 10^5 \quad k: 10^0$	1	2,4	1,1	1,1	5,3
$n: 10^5 \quad k: 10^1$	5,9	2,2	1	1	4,9
$n: 10^5 \quad k: 10^2$	67	2,1	1	1,1	4,8
$n: 10^5 \quad k: 10^3$	304	1	1,1	1,3	2,3
$n: 10^5 \quad k: 10^4$	1445	1	33,1	43,3	1,7
$n: 10^5 \quad k: 10^5$	∞	1	∞	∞	1,9
$n: 10^6 \quad k: 10^0$	1	3,9	1,2	1,3	8,1
$n: 10^6 \quad k: 10^1$	6,6	2,7	1	1	7,3
$n: 10^6 \quad k: 10^2$	83,1	3,2	1	1,1	6,6
$n: 10^6 \quad k: 10^3$	690	2,2	1	1,1	5,7
$n: 10^6 \quad k: 10^4$	∞	1	5	6,4	1,9
$n: 10^6 \quad k: 10^5$	∞	1	∞	∞	1,7
$n: 10^6 \quad k: 10^6$	∞	1	∞	∞	1,8
$n: 10^7 \quad k: 10^0$	1	3,4	1,1	1,1	7,4
$n: 10^7 \quad k: 10^1$	8,6	2,6	1	1,1	6,7
$n: 10^7 \quad k: 10^2$	82,1	2,6	1	1,1	6,8
$n: 10^7 \quad k: 10^3$	∞	3,1	1	1,1	6,6
$n: 10^7 \quad k: 10^4$	∞	1,1	1	1,2	2,6
$n: 10^7 \quad k: 10^5$	∞	1	∞	∞	2,2
$n: 10^7 \quad k: 10^6$	∞	1	∞	∞	1,2
$n: 10^7 \quad k: 10^7$	∞	1	∞	∞	1,7
	Seleção	Quicksort	Inserção	Inserção2	Heapsort

Fonte:http://www2.dcc.ufmg.br/livros/algoritmos-edicao2/cap4/transp/completo1/cap4.pdf

Comparação empírica dos algoritmos

- Para valores de k até 1.000, o método da **PartialInsertionSort** é imbatível.
- O **PartialQuicksort** nunca ficar muito longe da InserçãoParcial.
- Na medida em que o k cresce o **PartialQuicksort** é a melhor opção.
- Para valores grandes de k, o método da **PartialInsertionSort** se torna ruim.
- Um método indicado para qualquer situação é o **PartialQuicksort**.
- O **PartialHeapsort** tem comportamento parecido com o do **PartialQuicksort**.
- No entano o PartialHeapsort.

Fonte:http://www2.dcc.ufmg.br/livros/algoritmos-edicao2/cap4/transp/completo1/cap4.pdf

Atividade em laboratório

Atividade

(1) Criar uma área no Cloud9: **https://c9.io/**

(2) Baixar do repositório do Tidia o arquivo: **aed2-05.tgz**

(3) Subir o arquivo **aed2-05.tgz** a sua área no Cloud9

(4) Descomprimir o arquivo **aed2-05.tgz** (no Cloud9).

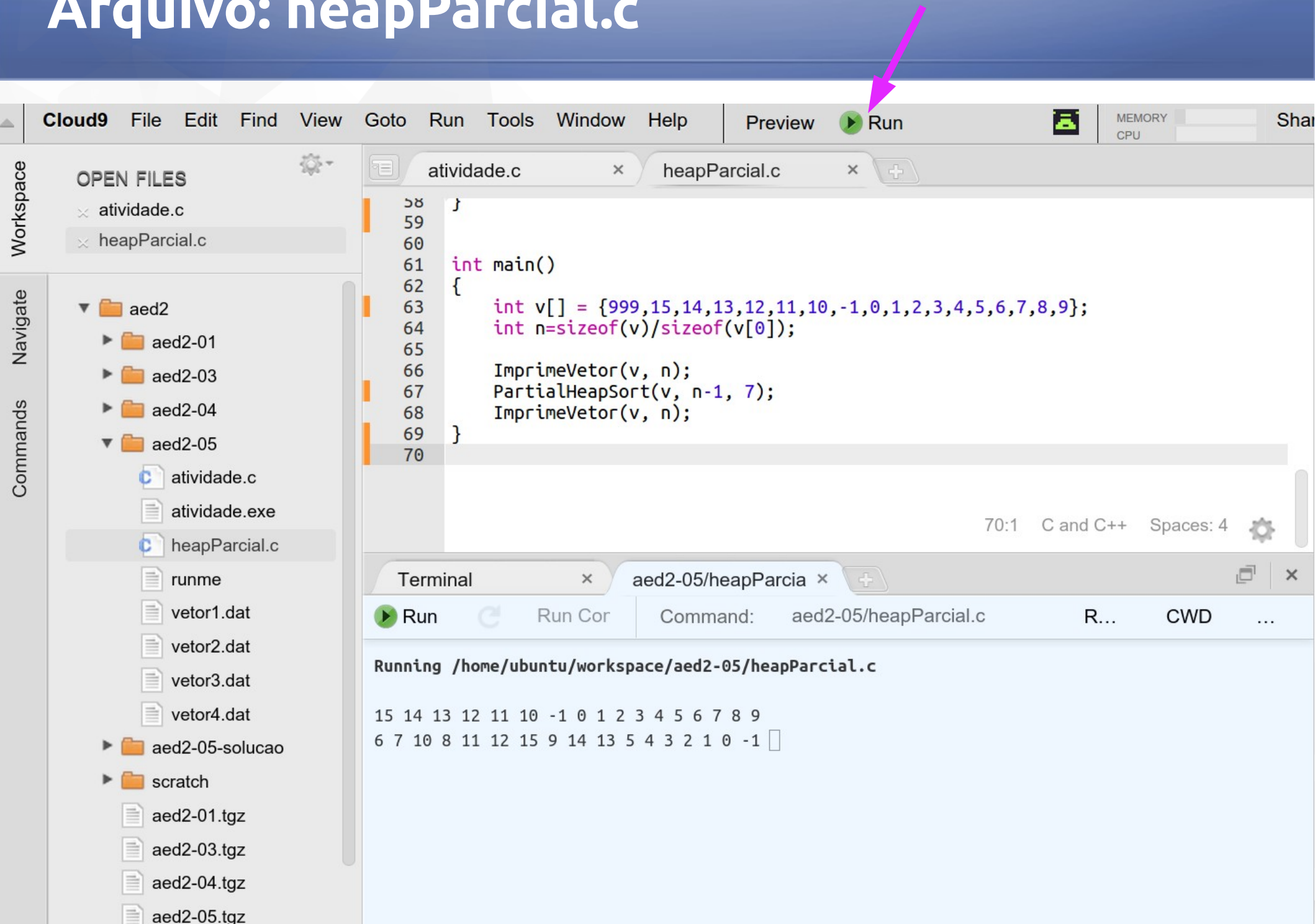
→ Abrir um terminal (Alt-T)

→ `tar xvfz aed2-05.tgz`

- Os arquivos da atividade estão no diretório aed2-05:

```
→ cd aed2-05
→ ls -lsh
→ cat /proc/cpuinfo
→ free -h
→ htop
```

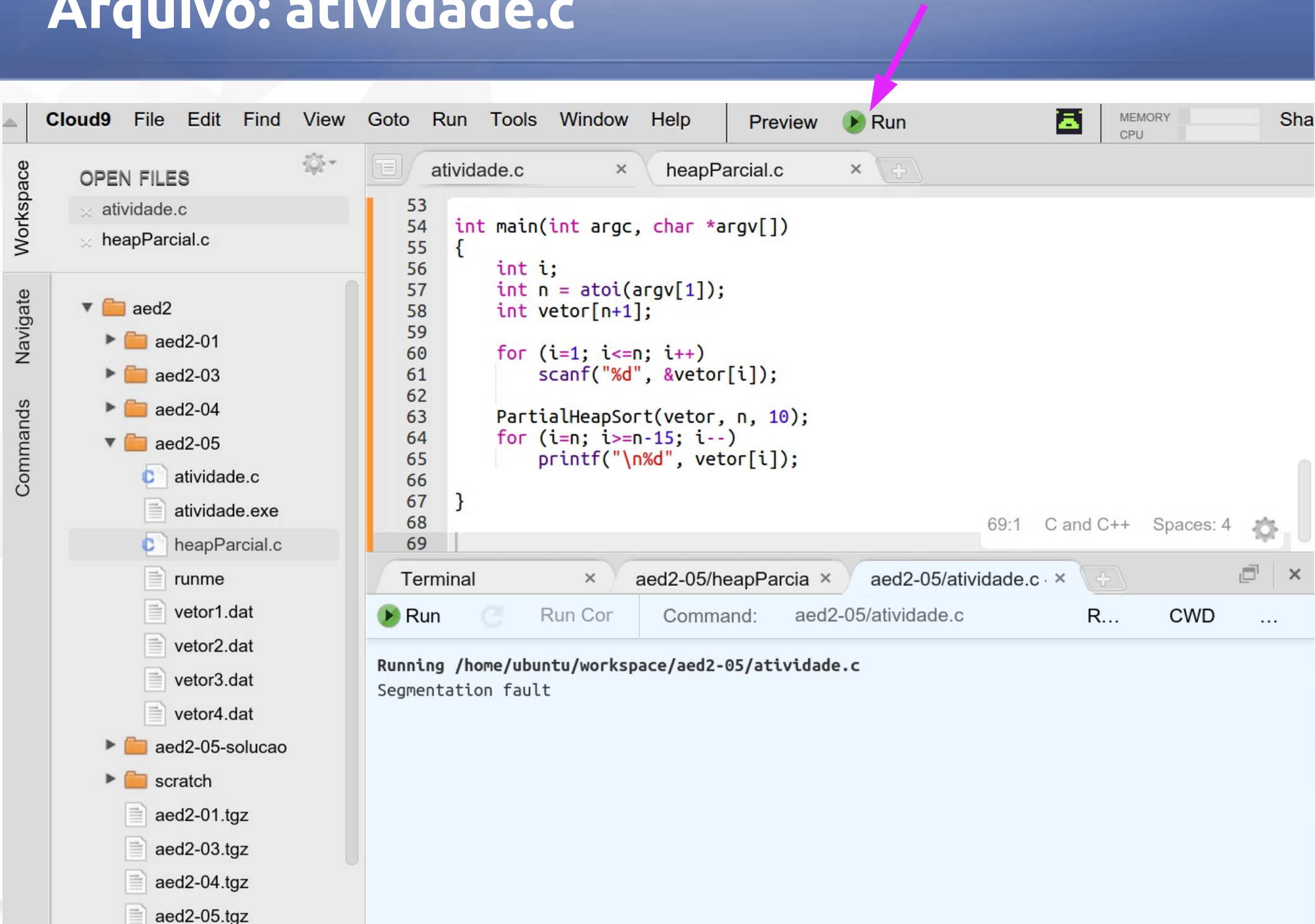
Arquivo: heapParcial.c



Arquivo: atividade.c

```c
int main(int argc, char *argv[])
{
    int i;
    int n = atoi(argv[1]);
    int vetor[n+1];

    for (i=1; i<=n; i++)
        scanf("%d", &vetor[i]);

    PartialHeapSort(vetor, n, 10);
    for (i=n; i>=n-15; i--)
        printf("\n%d", vetor[i]);

}
```

Arquivo: atividade.c



Arquivo: atividade.c

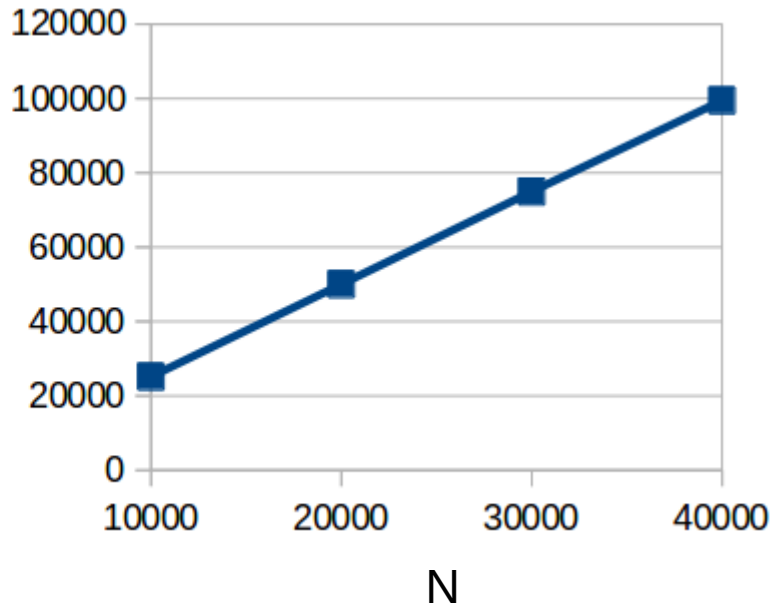
```
jmenac@aed2:/home/ubuntu/workspace/aed2-05 $ gcc atividade.c -o atividade.exe
jmenac@aed2:/home/ubuntu/workspace/aed2-05 $ ./atividade.exe 40000 < vetor4.dat

355
69021
163619
191664
263006
296648
323724
416314
452095
482273
1887852428
1495234472
1236034577
1615646145
1744558121
2100794804jmenac@aed2:/home/ubuntu/workspace/aed2-05 $
```

Arquivo: atividade.c

- Modifique o procedimento **PartialHeapSort** para que devolva o número de comparações realizadas para obter os k primeiros elementos do vetor dado como entrada.

```
int PartialHeapSort(int A[], int n, int k)
```

Arquivo	N	Comparações
vetor1.dat	10000	25144
vetor2.dat	20000	50014
vetor3.dat	30000	74948
vetor4.dat	40000	99442



Arquivo: atividade.c (solução)

```c
int MinHeapify (int A[], int m, int i) {
    int e, d, menor, aux;
    e = 2*i;
    d = 2*i+1;
    if (e<=m && A[e]<A[i])
        menor = e;
    else
        menor = i;
    if (d<=m && A[d]<A[menor])
        menor = d;

    if (menor!=i) {
        aux = A[menor];
        A[menor] = A[i];
        A[i] = aux;
        return 2+MinHeapify(A, m, menor);
    }
    else
        return 2;
}
```

```c
int PartialHeapSort(int A[], int n, int k) {
    int i, m, aux, comp=0;
    comp += BuildMinHeap(A, n);
    m = n;
    for (i=n; i>n-k; i--) {
        aux = A[i];
        A[i] = A[1];
        A[1] = aux;
        m = m-1;
        comp += MinHeapify (A, m, 1);
    }
    return comp;
}
```

```c
int BuildMinHeap(int A[], int n) {
    int i, comp=0;
    for (i=n/2; i>=1; i--)
        comp += MinHeapify (A, n, i);
    return comp;
}
```

Arquivo: atividade.c (solução)

```c
int main(int argc, char *argv[])
{
    int i;
    int n = atoi(argv[1]);
    int vetor[n+1];

    for (i=1; i<=n; i++)
        scanf("%d", &vetor[i]);

    printf("\ncomparacoes=%d", PartialHeapSort(vetor, n, 10) );

    for (i=n; i>=n-15; i--)
        printf("\n%d", vetor[i]);

}
```

Arquivo: atividade.c (solução)

```
jmenac@aed2:/home/ubuntu/workspace/aed2-05-solucao $ ./atividade.exe 40000 < vetor4.dat

comparacoes=99442
355
69021
163619
191664
263006
296648
323724
416314
452095
482273
1887852428
1495234472
1236034577
1615646145
1744558121
```

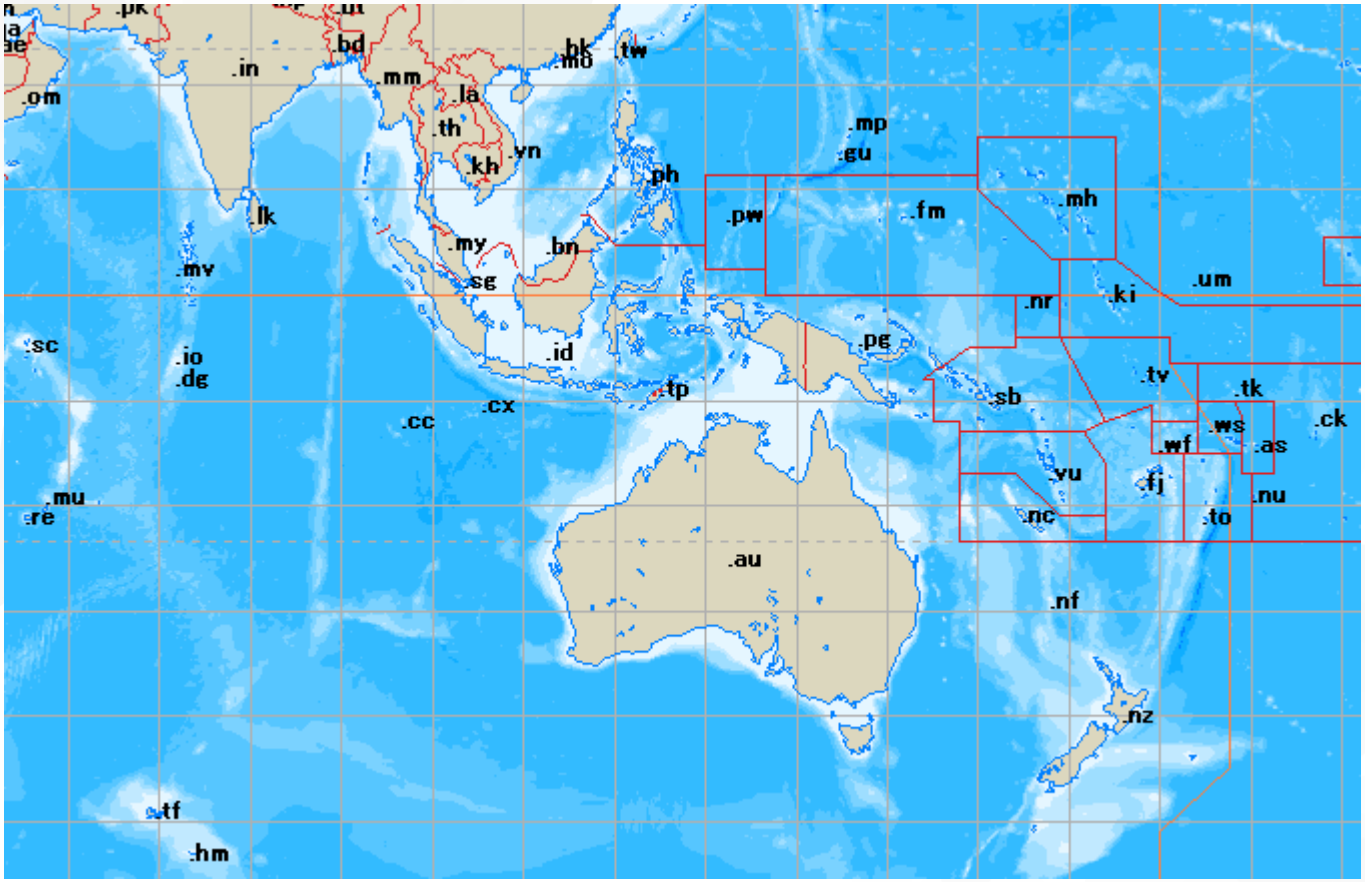
Arquivo: atividade.c (solução)

```
jmenac@aed2:/home/ubuntu/workspace/aed2-05-solucao $ ./atividade.exe 40000 < vetor4.dat

comparacoes=99442
355
69021
163619
191664
263006
296648
323724
416314
452095
482273
1887852428
1495234472
1236034577
1615646145
1744558121
```

```
jmenac@aed2:/home/ubuntu/workspace/aed2-05-solucao $ sort -n vetor4.dat | more
355
69021
163619
191664
263006
296648
323724
416314
452095
482273
488280
498967
632765
640349
647826
```

Para visualizar o manual do programa:
`$ man sort`

Domains: .io .fm .to .tv



Domains: .io .fm .to .tv

.fm	Federated States of Micronesia		Also unofficially used by FM radio stations or related business
.io	British Indian Ocean Territory		Used unofficially by technology companies/startups/web applications because IO can be an acronym for input/output which is useful for domain hacks.
.tv	Tuvalu		Used as an abbreviation of television, the domain is currently operated by dotTV, a VeriSign company; the Tuvalu government owns twenty percent of the company.
.to	Tonga		Often used unofficially for Torrent, Turin, Toronto, Tokyo, or Tocantins

Fonte: http://en.wikipedia.org/wiki/List_of_Internet_top-level_domains