

13	20	4	7	12	8	10
0	1	2	3	4	5	6

[illegible]

13	20	4	7	12	8	10
0	1	2	3	4	5	6

[illegible]

13	20	4	7	12	8	10
0	1	2	3	4	5	6

```

void insertionSort(int v[], int n) {
    int i, j, chave;
    for(j=1; j<n; j++) {
        chave = v[j];
        i = j-1;
        while(i >= 0 && v[i] > chave){
            v[i+1] = v[i];
            i--;
        }
        v[i+1] = chave;
    }
}

```

n=7

chave=

i	j	0	1	2	3	4	5	6

13	20	4	7	12	8	10
0	1	2	3	4	5	6

```

void ordena_por_contagem(int vet[], int ord[], int n){
    int i,j,p;
    //determinar a posição de cada elemento no vetor ordenado
    for(i=0;i<n;i++){
        p=0;
        for(j=0;j<n;j++){
            if (vet[i]>vet[j])
                p++;
        }
        ord[p]=vet[i];
    }
}

```

n=7

p

	i	j	0	1	2	3	4	5	6
vet									
ord									

13	20	4	7	12	8	10
0	1	2	3	4	5	6

```
void quicksort (int v[], int primeiro, int ultimo){//primeiro e último
índices do vetor
```

```
int i, j, m, aux;
i=primeiro; j=ultimo;
m=v[(i+j)/2];
do {
    while (v[i] < m) i++;
    while (v[j] > m) j--;
    if (i<=j){
        aux=v[i];
        v[i]=v[j];
        v[j]=aux;
        i++;
        j--;
    }
} while (i<=j);
if (primeiro < j)
    quicksort(v,primeiro,j); //chamada
recursiva da 1ª metade

    if (ultimo > i)
        quicksort(v,i,ultimo); //chamada recursiva da 2ª metade
}
```

primeiro=

ultimo=

i=

j=

m=

aux

i	j	0	1	2	3	4	5	6

13	20	4	7	12	8	10
0	1	2	3	4	5	6

```
void intercala (int p, int q, int r, int v[]) {
```

```
    int i, j, k, *w;
    w = (int*)malloc ((r-p) * sizeof
(int));
    i = p; j = q;
    k = 0;
    while (i < q && j < r) {
        if (v[i] <= v[j]) w[k++] = v[i++];
        else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}

void mergesort (int p, int r, int v[]) {
    if (p < r-1) {
        int q = (p + r)/2;
        mergesort (p, q, v);
        mergesort (q, r, v);
        intercala (p, q, r, v);
    }
}
```

p=0

r=6

q=3

	0	1	2	3		4	5	6	
	13	20	4	7		12	8	10	
13	20		4	7		8	12		10
	4	7	13	20		8	10	12	
	4	7	8	10	12	13	20		