O problema da seleção

Mediana de N *elementos:* elemento x tal que pelo menos $N/2$ dos elementos são $\geq x$ e pelo menos $N/2$ dos elementos são $\leq x$.

Problema 1. *Dado* N *itens, encontrar uma mediana.*

Mais geral:

Problema 2. *Dado* N *itens e* $1 \leq K \leq N$*, encontrar o* K*-ésimo menor elemento.*

O caso $K = \lceil N/2 \rceil$ corresponde a encontrar a mediana.

Solução elementar

- Determine os K menores elementos usando as primeiras K fases do algoritmo de ordenação por seleção. Complexidade: $\Theta(NK)$
- Ordene os N elementos e encontre o K-ésimo menor elemento. Complexidade: igual à complexidade do algoritmo de ordenação usado; pode chegar a ser $O(N \log N)$. *Quicksort* garante $O(N \log N)$ na média.

Solução baseada no algoritmo de partição do quicksort

- Basicamente baseado na noção de *divisão-e-conquista*
- Complexidade de tempo no caso médio: $O(N)$
- Pior caso: $\Theta(N^2)$
- Componente básico: `partition()`
  - Veremos futuramente um método que tem complexidade $O(N \log K)$ no pior caso.

Solução baseada no algoritmo de partição do quicksort

```
void select(Item a[], int l, int r, int k)
  { int i;
    if (r <= l) return;
    i = partition(a, l, r);
    if (i > k) select(a, l, i-1, k);
    if (i < k) select(a, i+1, r, k);
  }
```

▷ A chamada `select(a, l, r, k)` com `l <= k <= r` deixa o k-ésimo elemento em sua posição “correta”

Versão não recursiva

```
void select(Item a[], int l, int r, int k)
  {
    while (r > l)
      { int i = partition(a, l, r);
        if (i >= k) r = i-1;
        if (i <= k) l = i+1;
      }
  }
```

▷ Por que temos `i >= k` e não `i > k`? (Analogamente para `i <= k`.)
[Considere o caso `i = k`.]

Solução baseada no algoritmo de partição do quicksort

- Complexidade (para qualquer k): $O(N)$ no caso médio mas $\Omega(N^2)$ no pior caso

Problema 3. *Considere o problema da mediana* $(k = \lfloor (l + r)/2 \rfloor)$. *A complexidade de caso médio é basicamente da forma* cN *para uma constante* c. *Estime* c *experimentalmente.*

- Caso médio: entrada é uma permutação escolhida uniformemente ao acaso dentre as $N!$ possíveis

Desafio

Problema 4. *Projete um algoritmo para encontrar a mediana de* N *elementos com complexidade de tempo de pior caso* $O(N)$.

Mergesort: ordenação por intercalação

- Baseado na noção de *divisão-e-conquista* (divide-and-conquer)
- Complexidade de tempo no pior caso: $O(N \log N)$
- Gasta $O(N)$ de espaço extra (nas implementações "naturais")
- É *estável*
- Componente básico: algoritmo de intercalacão (*merge*)
- Além do Sedgewick (**sempre** leiam o Sedgewick), veja
  - `http://www.ime.usp.br/~pf/algoritmos/aulas/mrgsrt.html`

Algoritmo de intercalação: merging

```
/* prog8.1.c */
void mergeAB(Item c[], Item a[], int N, Item b[], int M)
  { int i, j, k;
    for (i = 0, j = 0, k = 0; k < N+M; k++)
      {
        if (i == N) { c[k] = b[j++]; continue; }
        if (j == M) { c[k] = a[i++]; continue; }
        c[k] = (less(a[i], b[j])) ? a[i++] : b[j++];
      }
  }
```

Algoritmo de intercalação: merging

Versão mais sofisticada (o maior elemento serve como sentinela):

```
/* prog8.2.c */
Item aux[maxN];
void merge(Item a[], int l, int m, int r)
  { int i, j, k;
    for (i = m+1; i > l; i--) aux[i-1] = a[i-1];
    for (j = m; j < r; j++) aux[r+m-j] = a[j+1];
    for (k = l; k <= r; k++)
       if (less(aux[i], aux[j]))
          a[k] = aux[i++]; else a[k] = aux[j--];
  }
```

Mergesort: ordenação por intercalação

```
/* prog8.3.c */
void mergesort(Item a[], int l, int r)
  { int m = (r+l)/2;
    if (r <= l) return;
    mergesort(a, l, m);
    mergesort(a, m+1, r);
    merge(a, l, m, r);
  }
```

Mergesort: ordenação por intercalação

Propriedade 5. *O número de comparações feito por mergesort só depende do número de elementos* N *na entrada. Este número é basicamente* $N \log_2 N$.

Seja $C(N)$ o número de comparações. Então

$$C(1) = 0 \quad \text{e} \quad C(N) = C(\lfloor N/2 \rfloor) + C(\lceil N/2 \rceil) + N \quad \text{para } N > 1. \qquad (1)$$

Esta recorrência pode ser resolvida explicitamente. Obtemos

$$C(N) = N\lceil \log_2 N \rceil + N - 2^{\lceil \log_2 N \rceil}. \qquad (2)$$

▷ Prove (2) para Ns da forma 2^k. Como você poderia tratar o caso genérico?

Mergesort: ordenação por intercalação

Propriedade 6. *Mergesort usa espaço extra proporcional a* N.

Propriedade 7. *Mergesort é estável.*