Mergesort: ordenação por intercalação

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

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);
  }
```

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: *top-down*, *bottom-up*

- `prog8.3.c`: versão *top-down*
- `prog8.5.c`: versão *bottom-up*

Mergesort: versão *bottom-up*

```
#define min(A, B) (A < B) ? A : B

void mergesortBU(Item a[], int l, int r)
  { int i, m;
    for (m = 1; m <= r-l; m = m+m)
      for (i = l; i <= r-m; i += m+m)
        merge(a, i, i+m-1, min(i+m+m-1, r));
  }
```

Mergesort: versão *bottom-up*

Uma *passada*: uma execução do corpo do laço externo de `mergesortBU()`

Propriedade 1. *Em cada passada do bottom-up mergesort, os subvetores intercalados têm tamanhos que são potências de* 2*, exceto, possivelmente, pelo último.*

Propriedade 2. *O número de passadas do bottom-up mergesort para se ordenar um vetor com* $N > 1$ *itens é igual ao número de bits na representação binária de* $N - 1$*.*

Por exemplo: para $N = 2$ ($N-1 = (1)_2$), ocorre exatamente uma passada; para $N = 21$ ($N - 1 = (10100)_2$), ocorrem exatamente 5 passadas.

Mergesort: versão otimizada

- Usamos ordenação por inserção para ordenar subvetores pequenos.
- Evitamos cópias desnecessárias de `a[]` em `aux[]`.

Mergesort: versão otimizada

```
/* prog8.4.c */
#define maxN 10000
Item aux[maxN];
void mergesortABr(Item a[], Item b[], int l, int r)
  { int m = (l+r)/2;
    if (r-l <= 10) { insertion(a, l, r); return; }
    mergesortABr(b, a, l, m);
    mergesortABr(b, a, m+1, r);
    mergeAB(a+l, b+l, m-l+1, b+m+1, r-m);
  }
void mergesortAB(Item a[], int l, int r)
  { int i;
    for (i = l; i <= r; i++) aux[i] = a[i];
    mergesortABr(a, aux, l, r);
  }
```

Mergesort: versão superotimizada

- Em `prog8.4.c`, usamos `mergeAB()` em vez de `merge()`.
- *Exercício*: escreva uma versão do mergesort que evita cópias redundantes dos dados (como é evitado em `mergesortAB()`), mas que também explora o uso de sentinelas como em `merge()` (isto não é feito em `mergesortAB()`, que usa `mergeAB()` e não `merge()`).

Mergesort para listas ligadas

```
/* prog8.6.c */
link merge(link a, link b)
  { struct node head; link c = &head;
    while ((a != NULL) && (b != NULL))
      if (less(a->item, b->item))
        { c->next = a; c = a; a = a->next; }
      else
        { c->next = b; c = b; b = b->next; }
    c->next = (a == NULL) ? b : a;
    return head.next;
  }
```

Mergesort para listas ligadas (top-down)

```
/* prog8.7.c */
link mergesort(link c)
  { link a, b;
    if (c->next == NULL) return c;
    a = c; b = c->next;
    while ((b != NULL) && (b->next != NULL))
      { c = c->next; b = b->next->next; }
    b = c->next; c->next = NULL;
    return merge(mergesort(a), mergesort(b));
  }
```

Mergesort para listas ligadas (bottom-up)

```
/* prog8.8.c */
link mergesort(link t)
  { link u;
    for (QUEUEinit(); t != NULL; t = u)
      { u = t->next; t->next = NULL; QUEUEput(t); }
    t = QUEUEget();
    while (!QUEUEempty())
      { QUEUEput(t); t = merge(QUEUEget(), QUEUEget()); }
    return t;
  }
```

Mergesort para listas ligadas

Exercício 3. *Compare a eficiência de tempo de* `prog8.7c`, `prog8.8.c`, *e* `prog3.11.c` *(ordenação de listas por inserção).*