Quicksort

Além do Sedgewick (**sempre** leiam o Sedgewick), veja

- `http://www.ime.usp.br/~pf/algoritmos/aulas/quick.html`

Quicksort

- Baseado na noção de *divisão-e-conquista* (divide-and-conquer)
- Complexidade de tempo no caso médio: $O(N \log N)$
- Pior caso: $\Theta(N^2)$
- Componente básico: `partition()`

Quicksort

- O algoritmo de partição do quicksort rearranja o vetor dado `a[l..r]` e determina um índice `i` de forma que
  - o objeto em `a[i]` está em sua posição final
  - todos os elementos em `a[l..i-1]` são menores ou iguais a `a[i]`
  - todos os elementos em `a[i+1..r]` são maiores ou iguais a `a[i]`
- Suponha que `partition(a, l, r)` executa tal rearranjo e devolve o índice `i`

Quicksort

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

Quicksort: algoritmo de partição

```
int partition(Item a[], int l, int r)
  { int i = l-1, j = r; Item v = a[r];
    for (;;)
      { while (less(a[++i], v)) ;
        while (less(v, a[--j])) if (j == l) break;
        if (i >= j) break;
        exch(a[i], a[j]);
      }
    exch(a[i], a[r]);
    return i;
  }
```

Quicksort

```
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ prog7.1b 16 1
84 39 78 79 91 19 33 76 27 55 47 62 36 51 95 91
[l r i =  0 15 13] 84 39 78 79 51 19 33 76 27 55 47 62 36 91 95 91
[l r i =  0 12  3] 27 33 19 36 51 78 39 76 84 55 47 62 79 91 95 91
[l r i =  0  2  0] 19 33 27 36 51 78 39 76 84 55 47 62 79 91 95 91
[l r i =  1  2  1] 19 27 33 36 51 78 39 76 84 55 47 62 79 91 95 91
[l r i =  4 12 11] 19 27 33 36 51 78 39 76 62 55 47 79 84 91 95 91
[l r i =  4 10  5] 19 27 33 36 39 47 51 76 62 55 78 79 84 91 95 91
[l r i =  6 10 10] 19 27 33 36 39 47 51 76 62 55 78 79 84 91 95 91
[l r i =  6  9  7] 19 27 33 36 39 47 51 55 62 76 78 79 84 91 95 91
[l r i =  8  9  9] 19 27 33 36 39 47 51 55 62 76 78 79 84 91 95 91
[l r i = 14 15 14] 19 27 33 36 39 47 51 55 62 76 78 79 84 91 91 95
19 27 33 36 39 47 51 55 62 76 78 79 84 91 91 95
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$
```

Quicksort não-recursivo

```
/* prog7.3.c */
#define push2(A, B)  STACKpush(B); STACKpush(A);
void quicksort(Item a[], int l, int r)
  { int i;
    STACKinit(); push2(l, r);
    while (!STACKempty())
      { l = STACKpop(); r = STACKpop();
        if (r <= l) continue;
        i = partition(a, l, r);
        push2(l, i-1); push2(i+1, r);
      }
  }
```

Tamanho da pilha?

- Tipicamente $O(\log N)$
- Pior caso: $\Theta(N)$
- É possível garantir que a pilha tenha no máximo $1+\log_2 N$ elementos ao ordenarmos N objetos, adotando uma política específica sobre a ordem de execução dos `push2()`

Quicksort não-recursivo criterioso

```
/* prog7.3.c */
#define push2(A, B)  STACKpush(B); STACKpush(A);
void quicksort(int a[], int l, int r)
  { int i;
    STACKinit(); push2(l, r);
    while (!STACKempty())
      { l = STACKpop(); r = STACKpop();
        if (r <= l) continue;
        i = partition(a, l, r);
        if (i-l > r-i)
          { push2(l, i-1); push2(i+1, r); }
        else
          { push2(i+1, r); push2(l, i-1); }
      }
  }
```

Quicksort não-recursivo criterioso

Propriedade 1. *Colocando o intervalo maior na pilha antes o intervalo menor (como no* `prog7.3.c`*), a pilha nunca contém mais que* $1 + \log_2 N$ *elementos ao ordenar* $N > 0$ *objetos.*

Prova. O tamanho máximo da pilha T_N obedece à recorrência $T_N \leq T_{\lfloor N/2 \rfloor} + 1$ ($N > 1$) e $T_1 = 1$. □

Árvore de recursão

- Ilustrativo: árvore de recursão de `prog7.1b 16 1`
- Árvores binárias têm muitos nós externos
- *Idéia de cutoff*: em vez de terminar a recursão em $N \leq 1$, podemos cortar a recursão antes, e depois executar insertion sort
- Mais uma idéia: *median-of-three* partitioning

Quicksort com median of three + cutoff

```
/* prog7.4.c */
#define M 10
void quicksort(Item a[], int l, int r)
  { int i;
    if (r-l <= M) return;
    exch(a[(l+r)/2], a[r-1]);
    compexch(a[l], a[r-1]);
      compexch(a[l], a[r]);
        compexch(a[r-1], a[r]);
    i = partition(a, l+1, r-1);
    quicksort(a, l, i-1);
    quicksort(a, i+1, r);
  }
```

Quicksort com median of three + cutoff

```
/* prog7.4.c */
#define M 10
[...]
void sort(Item a[], int l, int r)
  {
    quicksort(a, l, r);
    insertion(a, l, r);
  }
```

Quicksort: testes para strings?

- `prog3.17a.c`: ordenação de palavras; usa insertion sort. Escreva versão alternativa com quicksort. Execute em arquivos grandes. Por exemplo, textos do Projeto Gutenberg.