__/__/__

Quicksort

```
fn QuickSort(p, q) {
    // Sorts a[p], ..., a[q] which are in an array a[0..n-1],
    // a[n] is considered to exist and ≥ all elements in a[0..n-1].
    if (p<q) then {
        j = Partition(a, p, q+1);
        QuickSort(p, j-1);
        QuickSort(j+1, q);
    }
}

fn Partition(a, m, p) {
    // Rearrange the elements in such a way that the contents
    // of a[m] are shifted alongwith others may to bring all
    // elements less than it before itself and all ~~those~~ after.
    // Strictly speaking, if initially t = a[m], then after completion
    // a[q] = t for some q between m and p-1, a[k] ≤ t for
    // m ≤ k < q and a[k] ≥ t for q < k < p.
    // fn returns q.
    v = a[m]; i = m; j = p;
    repeat {
        repeat i = i+1 until (a[i] ≥ v);
        repeat j = j-1 until (a[j] ≤ v);
        if (i<j) then interchange(a[i], a[j]);
    } until (i ≥ j);
    a[m] = a[j]; a[j] = v;
    return j;
}
```

__/__/__

Analysis:

Let $C(n)$ be the element comparisons in QuickSort.
We assume that the n elements to be sorted are distinct and the input distⁿ is such that the partition element $v = a[m]$ in the call to Partition(a, m, p) has an equal probability of being the i^{th} smallest element, $1 \leq i \leq p-m$ in $a[m..p-1]$.

The number of element comparisons in each call of Partition is at most $p-m+1$. Let r be the total number of elements in all the calls to Partition at any level of recursion. At level one only one call, Partition$(a, 0, n)$ is made and $r = n-1$; at level two at most two calls are made and $r = n-2$ and so on. At each level of recursion, $O(r)$ element comparisons are made by Partition. At each level, r is at least one less than the r at the previous level as the partitioning elements of the previous level are eliminated. Hence, the worst-case comparisons,

$$C_w(n) = \sum_{1}^{n-1} r = O(n^2)..$$

The average case $C_A(n)$ of $C(n)$ is much less than $C_w(n)$. Under the assumptions made, the partitioning element v has an equal probability of being the i^{th}-smallest element, $1 \leq i \leq p-m$, in $a[m..p-1]$. Hence the two subarrays remaining to be sorted are $a[m:j]$ and $a[j+1:p-1]$ with probability $1/(p-m)$, $m \leq j < p$. From this we obtain the recurrence relation

$$C_A(n) = n-1+1+\frac{1}{n-1}\sum_{0\le k\le n-1}\left[C_A(k-1)+C_A(n-1-k)\right]$$

The number of element comparisons required by Partition on its first call is n. Note that $C_A(0)=C_A(1)=0$.

Multiplying both sides by n,

$$n\,C_A(n) = n(n+1) + 2\left[C_A(0)+C_A(1)+\cdots+C_A(n)\right]$$

Replacing n by $n-1$

$$(n-1)C_A(n-1) = n(n-1) + 2\left[C_A(0)+\cdots+C_A(n-2)\right]$$

Subtracting this from the last eqn, we get

$$nC_A(n) - (n-1)C_A(n-1) = 2n + 2C_A(n-1)$$

or $$\frac{C_A(n)}{n+1} = \frac{C_A(n-1)}{n} + \frac{2}{n+1}$$

Repeatedly using this eqn. to substitute for $C_A(n-1), C_A(n-2), \ldots$ we get

$$\frac{C_A(n)}{n+1} = \frac{C_A(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \frac{C_A(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$\vdots$$

$$= \frac{C_A(1)}{2} + 2\sum_{3\le k\le n+1} 1/k$$

$$= 2\sum_{3\le k\le n+1} 1/k$$

Since

$$\sum_{3 \le k \le n+1} 1/k \le \int_2^{n+1} \frac{1}{x} dx = \log_e (n+1) - \log_e 2,$$

we get

$$C_A(n) \le 2(n+1)\left[\log_e (n+2) - \log_e 2\right] = O(n \log n).$$

Randomized Quicksort

Instead of picking up a[n] as the pivot, we propose to choose, rather pick a random element as the partition element or pivot. The resultant randomized algorithm:

```
fn RQuickSort(p, q){
    // Sorts the elements a[p],.., a[q] which are in an array
    // a[0..n-1], a[n] is considered to be defined and ≥ all elements
    // in a[0..n-1].
    if (p<q) then {
        if ((q-p)>5) then
            interchange(a, Random() mod(q-p+1) +p, p);
        j = Partition(a, p, q+1);
        RQuickSort(p, j-1);
        RQuickSort(j+1, q);
    }
}
```

This is a Las Vegas algorithm since it will always output the correct answer.

__/__/____

Every call to the randomizer Random takes a certain amount of time. If there are only a few elements to sort, the time taken by the randomizer may be comparable to the rest of the computation. That is why, we invoke the randomizer only if $(q-p) > 5$, of course, 5 chosen empirically.

Let $A(n)$ be the average time of RQuickSort on any input of n elements. The recurrence relation for $A(n)$ is

$$A(n) = \frac{1}{n} \sum_{1 \le k \le n} \left(A(k-1) + A(n-k)\right) + n + 1.$$

Like before,

$$A(n) = O(n \log n).$$