QuickSort

Median

S = set of n numbers.

x is the (lower) median of S if x is the $\lfloor (n+1)/2 \rfloor$'th smallest element of S.

(x is the $\lfloor (n+1)/2 \rfloor$'th element of S when S is sorted in increasing order.)

Median: Example

S = set of n numbers.

x is the (lower) median of S if x is the $\lfloor (n+1)/2 \rfloor$'th smallest element of S.

(x is the $\lfloor (n+1)/2 \rfloor$'th element of S when S is sorted in increasing order.)

What is the median of the following 11 elements?

$$4, 17, 7, 25, 15, 9, 20, 14, 12, 2, 19$$

Sort 2: Divide and Conquer

Input : Array A of at least j elements.

Integers i and j.

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

```
Sort2(A[ ],i,j)
if (i < j) then
    p ← Median(A, i, j);
    /* Partition A[i, ..., j] using p s.t. A[s] = p          */
    /* and A[i'] ≤ p ≤ A[j'] for i ≤ i' ≤ s ≤ j' ≤ j           */
    s ← Partition(A, i, j, p);
    Sort2(A[ ],i,s − 1);
    Sort2(A[ ],s + 1,j);
end
```

`Sort2`: Example

Apply `Sort2` to the following 11 elements:

$$4, 17, 7, 25, 15, 9, 20, 14, 12, 2, 19$$

Simple Partition

Input : Array A of at n elements. Array element p.

Result : A permutation of array A such that $\mathsf{A}[s] = p$ and $\mathsf{A}[i'] \leq p \leq A[j']$ for $i \leq i' \leq s \leq j' \leq j$. Returns s.

```
SimplePartition(A[ ], n, p)
for m = 1 to n do
    if (A[m] < p) then Add A[m] to the end of array B;
    else if (A[m] > p) then Add A[m] to the end of array C;
    else
        Add A[m] to the end of array C;
        Swap the first and last element of array C;
    end
end
Replace A with the elements of B followed by the elements of C;
return (|B|);             /* |B| is the number of elements in B */
```

Simple Partition: Example

`SimplePartition` $([4, 17, 7, 25, 15, 9, 20, 14, 12, 2, 19],\ 14)$

Simple Partition: Running Time

Input : Array A of at n elements. Array element p.

Result : A permutation of array A such that $\mathsf{A}[s] = p$ and $\mathsf{A}[i'] \le p \le A[j']$ for $i \le i' \le s \le j' \le j$. Returns s.

```
SimplePartition(A[ ], n, p)
for m = 1 to n do
    if (A[m] < p) then Add A[m] to the end of array B;
    else if (A[m] > p) then Add A[m] to the end of array C;
    else
        Add A[m] to the end of array C;
        Swap the first and last element of array C;
    end
end
Replace A with the elements of B followed by the elements of C;
return (|B|);              /* |B| is the number of elements in B */
```

Partition

Input : Array A of at least j elements.
Integers i and j. Array element p.

Result : A permutation of array A such that $\mathsf{A}[s] = p$ and $\mathsf{A}[i'] \leq p \leq A[j']$ for $i \leq i' \leq s \leq j' \leq j$. Returns s.

```
Partition(A[ ], i, j, p)
low ← i;
high ← j;
while (low < high) do
    if (A[low] < p) then low ← low + 1;
    else if (A[high] > p) then high ← high − 1;
    else
        Swap(A[low], A[high]);
        if (A[low] = p and A[high] = p) then low ← low + 1;
    end
end
return (high);
```

Partition: Example

`Partition` $([4, 17, 7, 25, 15, 9, 20, 14, 12, 2, 19],\ 14)$

Partition: Running Time

Input : Array A of at least j elements.
Integers i and j. Array element p.

Result : A permutation of array A such that $\mathsf{A}[s] = p$ and $\mathsf{A}[i'] \leq p \leq A[j']$ for $i \leq i' \leq s \leq j' \leq j$. Returns s.

```
Partition(A[ ], i, j, p)
low ← i;
high ← j;
while (low < high) do
    if (A[low] < p) then low ← low + 1;
    else if (A[high] > p) then high ← high − 1;
    else
        Swap(A[low], A[high]);
        if (A[low] = p and A[high] = p) then low ← low + 1;
    end
end
return (high);
```

Sort 2: Running Time

Input : Array A of at least j elements.

Integers i and j.

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

```
Sort2(A[ ],i,j)
if (i < j) then
    p ← Median(A, i, j);
    /* Partition A[i,...,j] using p s.t. A[s] = p                */
    /* and A[i'] ≤ p ≤ A[j'] for i ≤ i' ≤ s ≤ j' ≤ j              */
    s ← Partition(A, i, j, p);
    Sort2(A[ ],i,s − 1);
    Sort2(A[ ],s + 1,j);
end
```

QuickSort

Input : Array A of at least j elements.
Integers i and j.

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

```
QuickSort(A[ ],i,j)
if (i < j) then
    /* choose random element of A[ ]                                   */
    p ← RandomElement(A, i, j);
    /* Partition A[i, ..., j] using p s.t. A[s] = p                    */
    /* and A[i'] ≤ p ≤ A[j'] for i ≤ i' ≤ s ≤ j' ≤ j                   */
    s ← Partition(A, i, j, p);
    QuickSort(A[ ],i,s − 1);
    QuickSort(A[ ],s + 1,j);
end
```

`QuickSort`: Example

Apply `QuickSort` to the following 11 elements:

$$4, 17, 7, 25, 15, 9, 20, 14, 12, 2, 19$$

QuickSort: Analysis

X_n = running time of `QuickSort` on array of size n.

$ET(X_n) = E(X_n)$ = expected running time of `QuickSort` on array of size n.

$ET(0) = 0$.

Assume array has no duplicates.

After partition, $A[s] = p$.

Let $m = s - i + 1$.
After partition, p is m'th element of $A[i], A[i+1], \ldots, A[j]$.

$$\begin{aligned} ET(n) &= \sum_{q=1}^{n} Pr(m=q)E(X_n|m=q) \\ &= \sum_{q=1}^{n} \frac{1}{n}(ET(q-1) + ET(n-q) + cn). \end{aligned}$$

QuickSort: Upper Bounds

X_n = running time of `QuickSort` on array of size n.

$ET(X_n) = E(X_n)$ = expected running time of `QuickSort` on array of size n.

Assume array has no duplicates.

Let $m = s - i + 1$. (After partition, p is m'th element of $A[i], \ldots, A[j]$.)

$$\begin{aligned}
ET(n) = {} & Pr(m < n/4)E(X_n|m < n/4)+ \\
& Pr(n/4 \leq m \leq 3n/4)E(X_n|n/4 \leq m \leq 3n/4)+ \\
& Pr(m > 3n/4)E(X_n|m > 3n/4).
\end{aligned}$$

$$\begin{aligned}
Pr(m < n/4) &= 1/4. \\
Pr(m > 3n/4) &= 1/4. \\
Pr(n/4 \leq m \leq 3n/4) &= 1/2. \\
ET(X_n|m < n/4) &\leq ET(X_n|m = 1) = cn + ET(n-1). \\
ET(X_n|m > 3n/4) &\leq ET(X_n|m = n) = cn + ET(n-1). \\
ET(X_n|n/4 \leq m \leq 3n/4) &\leq ET(X_n|m = n/4) = cn + ET(n/4) + ET(3n/4).
\end{aligned}$$

QuickSort: Upper Bounds

Let $m = s - i + 1$. (After partition, p is m'th element of $A[i], \ldots, A[j]$.)

$$Pr(m < n/4) = 1/4.$$

$$Pr(m > 3n/4) = 1/4.$$

$$Pr(n/4 \leq m \leq 3n/4) = 1/2.$$

$$ET(X_n | m < n/4) \leq ET(X_n | m = 1) = cn + ET(n-1).$$

$$ET(X_n | m > 3n/4) \leq ET(X_n | m = n) = cn + ET(n-1).$$

$$ET(X_n | n/4 \leq m \leq 3n/4) \leq ET(X_n | m = n/4) = cn + ET(n/4) + ET(3n/4).$$

$$\begin{aligned} ET(n) = {} & Pr(m < n/4) ET(X_n | m < n/4) + \\ & Pr(n/4 \leq m \leq 3n/4) ET(X_n | n/4 \leq m \leq 3n/4) + \\ & Pr(m > 3n/4) ET(X_n | m > 3n/4) \\ \leq {} & \frac{1}{4}\Big(cn + ET(n-1)\Big) + \frac{1}{2}\Big(cn + ET(n/4) + ET(3n/4)\Big) + \\ & \frac{1}{4}\Big(cn + ET(n-1)\Big). \end{aligned}$$

QuickSort: Upper Bounds

$$
\begin{aligned}
ET(n) &\leq \frac{1}{4}\Big(cn + ET(n-1)\Big) + \frac{1}{2}\Big(cn + ET(n/4) + ET(3n/4)\Big) + \\
&\quad \frac{1}{4}\Big(cn + ET(n-1)\Big) \\
&= cn + \frac{1}{2}ET(n-1) + \frac{1}{2}\Big(ET(n/4) + ET(3n/4)\Big) \\
&\leq cn + \frac{1}{2}ET(n) + \frac{1}{2}\Big(ET(n/4) + ET(3n/4)\Big). \\
\frac{1}{2}ET(n) &\leq cn + \frac{1}{2}\Big(ET(n/4) + ET(3n/4)\Big). \\
ET(n) &\leq 2cn + ET(n/4) + ET(3n/4) \\
&\leq c_2 n + ET(n/4) + ET(3n/4) \qquad \text{where } c_2 = 2c.
\end{aligned}
$$

$\therefore ET(n) \in O(n \log_2(n))$.

QuickSort: Lower Bounds

In the best case, p is the median.

$$\begin{aligned} ET(n) &\geq cn + ET(n/2) + ET(n/2) \\ &= cn + 2ET(n/2). \\ \therefore ET(n) &\in \Omega(n \log_2(n)). \end{aligned}$$

QuickSort: Version 2

Input : Array A of at least j elements.
Integers i and j.

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

```
QuickSort2(A[ ],i,j)
if (i < j) then
    p ← A[i];
    /* Partition A[i,...,j] using p s.t. A[s] = p                       */
    /* and A[i'] ≤ p ≤ A[j'] for i ≤ i' ≤ s ≤ j' ≤ j                    */
    s ← Partition(A, i, j, p);
    QuickSort2(A[ ],i,s − 1);
    QuickSort2(A[ ],s + 1,j);
end
```

`QuickSort2`: Analysis

$ET(n)$ = expected running time of `QuickSort2` on n values.
$ET(0) = 0$.

Assume array has no duplicates and
all permutations are equally likely.

$$\begin{aligned} ET(n) &= \sum_{k=1}^{n} Pr(s=k) ET(s=k) \\ &= \sum_{k=1}^{n} \frac{1}{n}(ET(k-1) + ET(n-k) + cn). \end{aligned}$$

`QuickSort`: Version 3

Input : Array A of at least j elements.
Integers i and j.

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

```
QuickSort3(A[ ],i,j)
if (i < j) then
    p ← median element of {A[i], A[⌊(i + j)/2⌋], A[j]};
    /* Partition A[i, ..., j] using p s.t. A[s] = p                 */
    /* and A[i'] ≤ p ≤ A[j'] for i ≤ i' ≤ s ≤ j' ≤ j                */
    s ← Partition(A, i, j, p);
    QuickSort3(A[ ],i,s − 1);
    QuickSort3(A[ ],s + 1,j);
end
```

QuickSort and Insertion Sort

Insertion Sort - Worst case running time - $\Theta(n^2)$

Insertion Sort - Expected running time - $\Theta(n^2)$

QuickSort - Worst case running time - $\Theta(n^2)$

QuickSort - Expected running time - $\Theta(n \log_2(n))$

However, for $n \leq 50$, Insertion Sort runs faster than QuickSort.

Good implementations of QuickSort switch to Insertion Sort when $n \leq 50$. (Note: QuickSort is recursive, so eventually $n \leq 50$ on all branches of the recursion tree.)