COMP20003
Algorithms and Data Structures
Quicksort

Nir Lipovetzky

Department of Computing and Information Systems

University of Melbourne

Semester 2

Quicksort

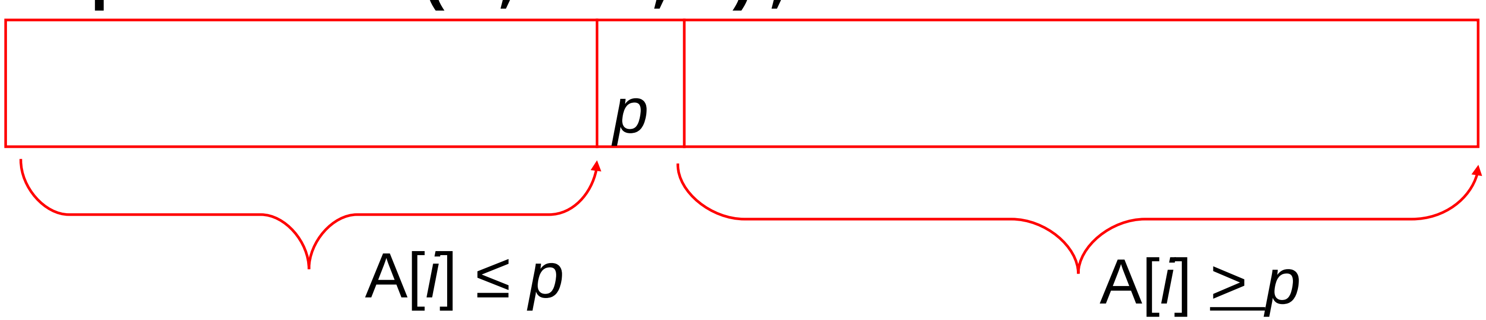
- A divide-and-conquer sorting algorithm.
- C.A.R. Hoare, “Quicksort”, *Computer Journal* **5**, 10-15, 1962.
- Skiena: Chapter 4.6

Quicksort: Basic idea

- Partition array:
  - Pick Pivot, which it is in its final position.
  - Everything larger than pivot has higher index.
  - Everything less than pivot has lower index.
- Recursion:
  - Partition left-half (recursively).
  - Partition right-half (recursively).
  - Base case: singletons are already sorted.

Quicksort code

```
int partition(item A[],int l,int r);
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: Concept *vs.* Implementation

- Conceptually simple.
- Partitioning does all the work.
- Partitioning is tricky.

```
/* call from quicksort(a,l,r) */
i = partition(a,l,r);

int partition(item A[], int l, int r)
{
      int i = l-1, j = r;
      item v = A[r];
      for( ; ; )
      {
         while (less(A[++i],v) /* do nothing */ ;
         while (less(v,A[--j]) /* do nothing */;
         if(i>=j) break;
         swap(A[i],A[j]);
      }
      swap(A[i],A[r]);
      return(i);
}



}
```

```
/* call from quicksort(a,l,r) */
i = partition(a,l,r);

int partition(item A[], int l, int r)
{
      int i = l-1, j = r;
      item v = A[r]; /* simplest, but NOT ideal */
      for( ; ; )
      {
         while (less(A[++i],v) /* do nothing */ ;
         while (less(v,A[--j]) /* do nothing */;
         if(i>=j) break;
         swap(A[i],A[j]);
      }
      swap(A[i],A[r]);
      return(i);
}



}
```

Quicksort

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Here they choose the pivot to be the

Left. Change algorithm slighly, last swap changes l and j, and initially i=l

Quicksort Exercize

15   10   13   27   12   22   20   25

Quicksort: analysis

- Best case:
- Worst case:
- Average case:

Quicksort inefficiencies

- Bad worst case for sorted or nearly sorted files.
- Fix:
  - Median-of-three or random partition element.

Quicksort inefficiencies

- Lots of function calls near the end for tiny subarrays.
- Fix:
  - Stop when r**`-l = SMALLNUMBER`**, and finish with **`XXXXsort.`**
  - Operationally, **`SMALLNUMBER ≈10`**

Quicksort summary: The Good the Bad and the Ugly

- The good:
  - Average case *n log n*.
  - In-place sort, no extra space required.
  - Inner loop is very quick

    (compare with mergesort).
  - Can be used in conjunction with other sorting algorithms, *e.g.* to make initial runs in multi-way mergesort from disk.

Quicksort summary: The Good the Bad and the Ugly

- The bad:
  - Worst case unlikely, but $O(n^2)$.
  - Ω (n log n) (even if file is already sorted).
  - Requires random access.

    Entire file must be in memory.

Quicksort summary: The Good the Bad and the Ugly

- The ugly:
  - Partition tricky to code.

http://www.youtube.com/watch?v=AFa1-kciCb4