Bucket Sort
and
Radix Sort

Juan Besa
Breaking the $n \log n$ barrier...

- $n \log n$ is optimal for comparisons...
- Today will see two algorithms that don’t use comparisons
  - Bucket Sort (or PigeonHole Sort)
  - Radix Sort
An important detail

• Because you are not comparing elements directly you need to have more information

• The most important information: What type are you sorting?
  • int, double, strings, points…

• But that is not enough
Bucket Sort

- You need to sort $n$ integers which lie in the range 1…$k$
• Use an array of size $k$ to store how many of each element there is

```c
sort(int n, X[n])
{
    int i, j, Y[k]
    for (i = 0; i < k; i++) Y[i] = 0;
    for (i = 0; i < n; i++) Y[X[i]]++;
    for (i = 0, j = 0; i < k; i++)
        while(Y[i]-- > 0) X[j++] = i
}
```
The complexity

• First loop takes $O(k)$

• Second and third loop take $O(n)$
  
  • Note that even though the third loop is nested it is looking at each element once and still $O(n)$

• Total time is $O(n+k)$ so this sort is very good when $k$ is smaller than $n$
Examples

• Sort 100000 by birthdays

• $k = 366$, $n = 100000$

• Total time is $O(n)$

• A bad example… sort 1000 people by their telephone numbers

• $k = 10^{10}$, $n = 100000$

• Total time is $O(n^2)$
Radix Sort

• What can we do when $k$ is large?

• An integer is a number of the form:
  • $a + 10 \times b + 100 \times c + \ldots$

• Where $a,b,c,\ldots$ are in the range 0-9

• This a range small enough to do a bucket sort!
The Code

- A list $L$ of Integers of the form: $a + 10 \times b + 100 \times c + \ldots$

```c
radix sort(L):
{
    bucket sort by a
    bucket sort by b
    bucket sort by c
    ...
}
```

Or

```c
radix sort(L):
{
    while (some key is nonzero)
    {
        bucket sort(keys mod 10)
        keys = keys / 10
    }
}
```
Why do the smaller digit first?

- If we do it by the largest digit we are separating the problem into subproblems
  - This is less efficient

- Radix sort doesn’t split up the list which tends to be more efficient

- Also maintains stability (if the bucket sort is stable)
Complexity

• Each bucket sort costs $O(n)$

• There are $\log_{10} k = O(\log n)$ bucket sorts, where $k$ is the largest number (like in bucket sort)

• So total time is $O(n \log k)$
Is this ever the best algorithm?

- If \( k < n \) then we could have used bucket sort and taken \( O(n) \) time

- If \( k > n \) then \( O(n \log k) > O(n \log n) \) we can obtain from comparison sorting
Implementation Details

• Using / and % 10 is slow, much better to use a power of 2 as base

• A large base is also better for example
  • If you are sorting 32 bit numbers and you use $2^{16}$ then you can do it in 2 bucket sort passes!

• With the correct implementation you can get very good speedups!
Where next?

- Radix sort can be used on strings
  - Lots of implementation details but very useful
- There are further limits but they tend to be unpractical