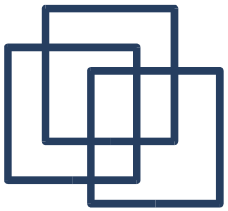
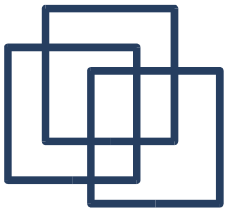
Sorting in Linear Time



Linear Sort? But...

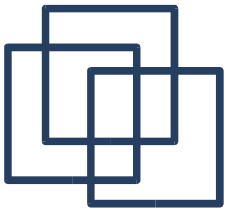
- Best algorithms so far perform in $\Theta(n\lg n)$. They are *comparison* sorts.
- Comparison sorts need at least $\Omega(n\lg n)$. Previous algorithms were *optimal*.
- Decision-tree model to prove $\Omega(n\lg n)$:
  - binary trees representing comparisons
  - all possible permutations are represented
  - $n!$ permutations for size n, thus, $n!$ leaves
  - sorting algorithms find an ordering, i.e., a path



Counting Sort

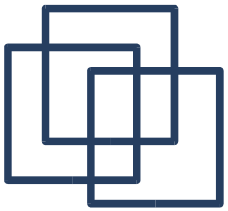
- Assume that $0 \le a_{1..n} \le k$. When $k=O(n)$, the sort runs in $\Theta(n)$ time.
- Idea: for every x, count how many elements are $\le x$, say t, then put x at t.
- **Count-sort(A,B,k):** // B is the output

```
for i:=0 to k do C[i]:=0                  // initialize
for i:=1 to length(A) do C[A[i]]++        // count is
for i:=1 to k do C[i]+=C[i-1]             // count ≤i
for i:=length[A] downto 1 do
    B[C[A[i]]]:=A[i]                      // write x at t
    C[A[i]]--                             // update counter
```



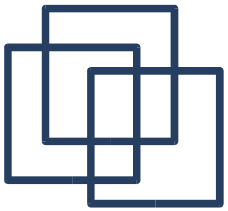
Counting Sort

- Running time in $\Theta(n+k)$, which becomes $\Theta(n)$ when $k=o(n)$.
- There is no comparison.
- The sort is *stable* (order kept for $a_i==a_j$).
- Problem: range of numbers that translate into the size of the working arrays.



Radix Sort

- Sort on digits of the numbers, *least significant digits first*, with a *stable* sort algorithm, i.e., counting sort.
- **Radix-sort(A,d):** // *d* digits
  for i:=1 **to** d **do** sort_stable A on digit i
- If we sort *n* *d*-digits numbers with each digit taking *k* values (i.e. base *k*), the running time is Θ(d(n+k)).
- Careful of the constants for comparison + it is not an in-place sorting algorithm.



Bucket Sort

- Assumes the input is uniformly distributed.
- Assumes the input in [0,1)
- **bucket-sort(A):**
  n:=length(A)
  for i:=1 **to** n **do** insert A[i] into list B[nA[i]]
  for i:=0 **to** n-1 **do** insertion_sort list B[i]
  concatenate lists B[0], B[1], ...,B[n-1]
- Running time: $T(n)=\Theta(n)+\sum_{i=0}^{n-1} O(n_i^2)$
- Expected running time: $\Theta(n)$.