Sorting in Linear Time

Comparison Sorts — $O(n \log n)$, $\Omega(n \log n)$ for some input
The best we can do for comparison sorts is $\Omega(n \log n)$.

Other techniques for sorting exist, such as Linear Sorting which is not based on comparisons. Linear Sorting techniques include:

Lower Bounds For Worst-Case Comparison Sorts

In general, assuming unique inputs, comparison sorts are expressed in terms of $a_i \leq a_j$ comparisons.

What is the best we can do on the worst case type of input?
What is the best worst-case running time?

Decision Tree Model

$n = 3$
Input: $<a_1, a_2, a_3>$
Number of possible outputs $= 3! = 6$
Analysis Of Decision Tree Model

Worst Case Comparisons is equal to height of decision tree

Lower bound on the worst case running time is the lower bound on the height of the decision tree.

Note that the number of leaves in the decision tree ≥ n!, where n = number elements in the input sequence.

A binary tree of height h has ≤ 2^h leaves.

n! ≤ 2^h

By Stirling Approximation n! = \sqrt{2\pi n}(n/e)^n(1 + \Theta(1/n))

Thus n! > (n/e)^n, h ≥ lg(n/e)^n = nlgn - nlge = \Theta(nlgn)

h ≥ lg(n!) ≥ \Theta(nlgn)
The best possible worst case running time for comparison sorts is thus $\Omega(n\log n)$.

Heapsort and Mergesort, both of which are $\Theta(n\log n)$, are asymptotically optimal.

---

**Counting Sort**

Assume algorithm is input $n$ elements ranging from 1 to $k$.


If $k = O(n)$, then $T(n)$ for counting sort = $\Theta(n)$.

---

**Example**

Let array $A$ be the following array ($n=8$, $k=7$).

```
1 ... n
5 7 3 5 2 1 4 3
```

Compute $C[j]$ as the number of elements in $A[] \leq j$. Array $C$ is thus the following array.

Click mouse to advance to next frame.

Array $B$ contains the final sorted array.

```
1 2 3 3 4 5 5 7
```
Pseudocode

CountingSort(A,B,k)  
for i = 1 to k  
    C[i] = 0  
for j = 1 to length(A)  
    C[A[j]] = C[A[j]] + 1  
for i = 2 to k  
    C[i] = C[i] + C[i-1]  
for j = length(A) downto 1  
    B[C[A[j]]] = A[j]  
    C[A[j]] = C[A[j]] - 1

Analysis

\[ T(n) = 5n + 4k \]
\[ = \Theta(n + k) \]
\[ = \Theta(n) \]
\[ \Omega(nlg n) \] does not apply because CountingSort is not a comparison sort.

Stable Sort

Two elements having the same value appear in the same order in the sorted sequence as they did in the input sequence.

Counting Sort is a Stable Sort.
Array A:

<table>
<thead>
<tr>
<th></th>
<th>5 (1)</th>
<th>7</th>
<th>3 (1)</th>
<th>5 (2)</th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>3 (2)</th>
</tr>
</thead>
</table>

Array B:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3 (1)</th>
<th>3 (2)</th>
<th>4</th>
<th>5 (1)</th>
<th>5 (2)</th>
<th>7</th>
</tr>
</thead>
</table>

If we change line from

for j = length(A) downto 1
to

for j = 1 to length(A)

Would the algorithm still sort? ________

Would the algorithm still be a stable sort? ________

Radix Sort

This sort was originally used to sort computer punch-card decks.

It is currently used for multi-key sorts (eg., year/month/day)

Consider each digit of the number as a separate key

Idea 1: Sort on most significant digit n, then sort on digit n-1, etc.

Problem: For old sorters, sort into 10 bins, but subsequent recursive sorts require all 10 bins. Operator must store the other 9 piles.

Idea 2: Sort on least significant digit, then sort on next least significant digit, etc.
Example

429  621  317  193
621  233  621  233
233  193  429  317
193  317  233  429
317  429  193  621

RadixSort(A,d)
   for i = 1 to d
      StableSort(A, digit(i))

Would the array always be sorted if we iterate from i=d down to 1?

Analysis

- If each digit is in the range 1 to k, use _______________
- Each pass over a digit is $\Theta(n + k)$
- For d digits $\Theta(dn + dk)$
- If $d$ is a constant and $k = O(n)$, $T(n) = ________$

Radix-n implies each digit can differentiate among $n$ different symbols. For example, in the previous case we assumed radix-10. This is why the name **Radix Sort** is given.
Accuracy

Why do later passes not mess up earlier sorts?
Prove that after pass p, sorted for digit p+1 to last digit
Prove by induction over p

• True for p=1. Apply Stable sort to digit 1, sorted after pass
• Assume true for p=i, prove true for p=i+1
• After pass i, compare two numbers x and y
• If x appears before y then one of following conditions must be true
  1. $x < y$ for digit $i$, then $x$ belongs before $y$ in sorted order
  2. $x = y$ for digit $i$, then $x < y$ for rest of number, placed in that
     order in earlier pass, not swapped because use stable sort

Example

Show how n integers in the range 1 to $n^2$ can be sorted in $O(n)$ time.

Use a Radix-n sort.
Each digit requires n symbols, and $\log_n n^2$ digits are needed ($d=2$, $k=n$).

\[
T(n) = \Theta(2n + 2k) \\
= \Theta(2n + 2n) \\
= \Theta(n)
\]
Bucket Sort

Bucket Sort assumes that the input values are _______________ distributed over the range [0,1), 0 ≤ x < 1.

Procedure:

- Divide inputs into n equal-sized subintervals (buckets) over the range [0,1).

- Sort buckets and concatenate the buckets.

- \( T(n) = \) ________

Example

BucketSort(A)  
\[
\begin{align*}
  n &= \text{length}(A) \\
  \text{for } i &= 1 \text{ to } n \\
  \quad \text{insert}(A[i], B[\text{floor}(nA[i])]) &= n \\
  \text{for } i &= 0 \text{ to } n-1 \\
  \quad \text{InsertionSort}(B[i]) &= n+1 \\
  \text{return}(B[0].B[1].....B[n-1]) &= n \\
\end{align*}
\]
Analysis

\[ T_{BS} = 4n + T_{IS}n + 3 \]

Let \( n_i \) = number of elements in bucket \( B[i] \).
The expected time to sort elements in \( B[i] = E(O(n_i^2)) = O(E(n_i^2)) \).

Note that the expected value of a random variable \( X \) as described in Appendix C, is

\[
E(X) = \sum_x x Pr(X = x)
\]

The variable \( x \) here represents the value of \( X \), and \( Pr(X=x) \) represents the probability that the value of \( X \) is \( x \).

The \[ ________ \] of a random variable \( X \) with mean \( E[X] \) is notated \( \text{Var}[X] \).

\[
\bullet E[X^2] = \text{Var}[X] + E^2[X]
\]

\[
\bullet P(X=k) = \binom{n}{k} p^k q^{n-k} \text{ This is the binominal distribution for } n \text{ trials.}
\]

Analysis

Thus, for all buckets,

\[
\sum_{i=0}^{n-1} O(E(n_i^2)) = O(\sum_{i=0}^{n-1} E(n_i^2)).
\]

Given:
- n elements, uniformly distributed over all possible values
- n buckets

What is the probability that an element will be inserted into some bucket i?

**Answer:**

Given n trials consisting of putting elements into buckets, how many elements will be inserted into bucket i?

\[ n_i = \text{Binomial}(n, p) \]

with mean \( E(n_i) = np = 1 \)

and variance \( Var(n_i) = np(1-p) = 1 - 1/n \)

\[ E(n_i^2) = 1 - 1/n + 1^2 \]

\[ = 2 - 1/n \]

\[ = \Theta(1) \]

Therefore, \( T_{IS} = \Theta(1) \)

and \( T_{BS} = 4n + \Theta(1)n + 3 = O(n) \) for the average case. In the worst case, the run time is \( n^2 \).

---

**Medians and Order Statistics**

The selection problem (assume distinct elements)

**Input:** Set A of n numbers and integer i such that \( 1 \leq i \leq n \).

**Output:** \( x \in A \) such that \( x \) is larger than \( i-1 \) elements of \( A \) (\( x \) is the ith smallest element).
\( i = 1: \quad \) \\
\( i = n: \quad \) \\
\( i = \frac{n+1}{2}: \quad \) 
\( \) (actually \( \lceil \frac{n+1}{2} \rceil \) and \( \lceil \frac{n+1}{2} \rceil \)).

**Simple Solution:** 
\[
\text{return}(A[i]).
\]
This is \( \) .

\[\]

**Min and Max**

**Min(A):** \( O(n) \) (\( \) comparisons)

**Max(A):** \( O(n) \) (\( \) comparisons)

**Both:** \( O(n) \) (\( \) comparisons)

\[\]

**Selection in Expected Linear Time**

Goal: Select \( i \)th smallest element from \( A[p..r] \).
Partition into \( A[p..q] \) and \( A[q+1..r] \)
If \( i \)th smallest element is in \( A[p..q] \)
then recurse on \( A[p..q] \)
else recurse on \( A[q+1..r] \)

Select(A, p, r, i)
1 if \( p = r \)
then return A[p]

else q = Partition(A, p, r)

k = q - p + 1

if i ≤ k

then return Select(A, p, q, i)

else return Select(A, q+1, r, i-k)

---

**Example**

Click mouse to advance to next frame.

---

**Analysis**

\[ T(n) = T(n-1) + \Theta(n) \] in the worst case

\[ = cn + c(n-1) + c(n-2) + ... + c \]

\[ = c \sum_{i=1}^{n} i = c \frac{n(n+1)}{2} = \Theta(n^2) \]

\[ T(n) = T(n/2) + \Theta(n) \] in the best case

\( a=1, b=2, f(n) = \Omega(n^{\log_b a + \epsilon}) \), \( \epsilon = 1 \)

Case 3: \( T(n) = \Theta(n) \)

Show: \( 1*f(n/2) \leq cf(n) \), \( c < 1 \)

\( f(n/2) \leq c*f(n) \)

\( c \geq 1/2 \)

---

**Random-Partition**

Random-Partition(A, p, r)
\[
i = \text{Random}(p, r)
\]
\[
\text{swap}(A[p], A[i])
\]
\[
\text{return}(\text{Partition}(A, p, r))
\]

\text{Random-Select}(A, p, r, i)
replace \text{Partition} with \text{Random-Partition}
replace \text{Select} with \text{Random-Select}

These are still \(\Theta(n^2)\) in the worst case
and \(O(n)\) in the average case.

However, we can get \(O(n)\) worst case performance with Select2.

\section*{Select2}

1. \(O(n)\): Divide \(n\) elements into \(\lfloor n/5 \rfloor\) groups of \________ elements
   and one group of \________ elements

2. \(O(n)\): Find \________ of each group.

3. \(T(\lfloor (n/5) \rfloor)\): Use Select2 recursively to find median of \(\lfloor n/5 \rfloor\) medians

4. \(O(n)\): Partition \________ elements around median into \________
   and \________ elements.

5. \(T(7n/10 + 6)\):
   If \(i \leq k\)
   then use Select2 to find \________ element in \________ lower elements
   else use Select2 to find \________ element in \________ higher elements
Example
Click mouse to advance to next frame.

Analysis

\[ T(n) \leq T(\left\lceil \frac{n}{5} \right\rceil) + T(7n/10 + 6) + O(n) \]

To get \( T(7n/10 + 6) \), note that the smallest partition size would be

\[ 3\left(\left\lfloor \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rfloor - 2 \right) \geq \left( \frac{3n}{10} \right) - 6 \]

\( 7n/10 + 6 \) denotes number of elements in larger partition

Assume \( T(n) \leq cn \) for \( \left\lfloor n/5 \right\rfloor \) and \( 7n/10+6 \)

\[ T(n) = O(n) \]

\[ T(n) \leq c(n/5) + c(7n/10 + 6) + O(n) \]

\[ = cn/5 + c7n/10 + c6 + O(n) \]

\[ = 1/10(2cn + 7cn + 60c + O(n)) \leq cn \]

\[ 2cn + 7cn + 60c + O(n) \leq 10cn \]

\[ 60c + O(n) \leq cn \]

\[ c(n - 60) \geq O(n) \]

\[ c \geq O(n) / (n - 60) \]

c is a valid constant for large enough n.

Stopping condition: \( T(n) = \Theta(1) \), if \( n \leq 80 \)