Bin Sort

- Sorting integers in Range \([1, \ldots, n]\)
- Add all elements to table and then
- Retrieve in order 1, 2, 3, \ldots, \(n\)
- “Stable” Sorting Method (repeated elements will end up in their original order)
Radix Sort

Input list
216 521 425 116 091 515 124 034 096 024

After sorting on least significant digit
521 091 124 034 024 425 515 216 116 096

After sorting on 2nd least significant digit
515 216 116 521 124 024 425 034 091 096

After sorting on 3rd least significant digit
024 034 091 096 116 124 216 425 515 521
• Sort \( n \) numbers whose values are in 
\([0,1,\ldots,n-1]\) can be done in \( O(n) \) time via bin sort (use \( n \) bins).

• Sort \( n \) numbers whose values are in 
\([0,1,\ldots,n^2 - 1]\) can be done on \( O(n) \) time via radix sort. Each key is \( \log n \) bits long. There are two keys per value.

\[
\underbrace{xxxx\ xxxx} \leftarrow \text{number in } [0,\ldots,n^2 - 1] \text{ have } 2 \log n \text{ bits.}
\]

• Sort \( n \) numbers whose values are in 
\([0,1,\ldots,n^k - 1]\) can be done in \( O(kn) \) time via radix sort. Each key is \( \log n \) bits long. There are \( k \) keys per value.

\[
\underbrace{xxxx\ xxxx\ \ldots\ xxxx} \leftarrow \text{number in } [0,\ldots,n^k - 1] \text{ have } k \log n \text{ bits.}
\]
Heap Sort

Sort $x_1, x_2, \ldots, x_n$

for $i=1$ to $n$
    insert $x_i$ in Heap
for $i=1$ to $n$
    delete min from Heap

Time complexity is $O(n \log n)$.
As we discussed before, the first phase (insert $n$ elements) can be done in $O(n)$ time. (bottom up)
Sorting method is Not Stable
Merge Sort

- [8][4][5][6][2][1][7][3] in array A
- [4, 8][5, 6][1, 2][3, 7] in array B, TC is $O(n)$
- [4, 5, 6, 8][1, 2, 3, 7] in array A, TC is $O(n)$
- [1, 2, 3, 4, 5, 6, 7, 8] in array B, TC is $O(n)$
- Number of passes is $\log_2 n$
- Time complexity is $T(n) = 2T(n/2) + cn$ when considering it top down. Solution to this recurrence relation is $O(n \log_2 n)$, and uses $O(n)$ additional space.
- Stable Sorting method
Merge Sort Analysis

Assume, \( n = 2^k \) for some positive integer \( k \)

\[
T(n) = \begin{cases} 
T(1) & n = 1 \\
2T(n/2) + cn & n > 1 
\end{cases}
\]

\[
T(n) \leq 2T(n/2) + cn \\
= 2(2T(n/4) + cn/2) + cn \\
= 4T(n/4) + 2cn \\
= 4(2T(n/8) + cn/4) + 2cn \\
= 8T(n/8) + 3cn \ldots \text{(Stop when } n/2^k = 1) \\
= 2^k T(1) + kcn \\
= n + cn \log n \]

\( O(n \log n) \)

if \( 2^k < n \leq 2^{k+1} \), \( T(n) \leq T(2^{k+1}) \leq T(2n) \), which is \( O(n \log n) \)
Quick Sort

template<class T> T
void QuickSort(T a[], int l, int r)
{ // Sort a[l:r], a[r+1] has large value.
    if (l >= r) return;
    int i = l, // left to right cursor
        j = r + 1; // right to left cursor
    T pivot = a[l]; // Partition wrt pivot
    while (true) {
        do { // find >= element on left side
            i = i + 1; } while (a[i] < pivot);
        do { // find <= element on right side
            j = j - 1; } while (a[j] > pivot);
        if (i >= j) break; // swap pair not found
        Swap(a[i], a[j]); }
    a[l] = a[j]; a[j] = pivot;
    QuickSort(a, l, j-1); // sort left segment
    QuickSort(a, j+1, r); // sort right segment
}
QuickSort

- A[r+1] has value larger than all other elements. Why?
- Time complexity bound is $O(n^2)$.
- $O(n)$ additional space (because of recursion)
- NONSTABLE Sorting method
QuickSort

- **Worst Case**
  
  \[
  \sum_{i=1}^{n} i \text{ which is } O(n^2)
  \]

- Many instances take \(\Omega(n^2)\)

- **NONSTABLE Sorting Method**

- **Space Complexity:** \((QS(n) = \text{QuickSort on } n \text{ elements})\)

\[
\Omega(n) \geq QS(n), QS(n - 1), QS(n - 2), \ldots, QS(1)
\]
Reduce Space Complexity

- Make procedure non-recursive
- Solve the smaller problem first
- for example
  \[
  \text{sort(1,80)} \\
  \Downarrow \text{size } n \\
  \text{sort(1,40), sort(42,80)} \\
  \Downarrow \text{size } \leq n/2 \\
  \text{sort(42,62), sort(64,80)} \\
  \Downarrow \text{size } \leq n/4 \\
  \text{sort(64,74), sort(76,80)} \\
  \vdots \\
  \rightarrow \log_2(n) \text{ space}
  \]
Quick Sort

Note: Changing the recursive program so that it calls the smallest subproblem first has space complexity $O(n)$.

$$QS(a[], l, r)$$

...  
if $j-1-1 <= r-(j+1)$
    then \begin{align*}
            &QS(a, l, j-1) \\
            &QS(a, j+1, r)
        \end{align*}
    else \begin{align*}
            &QS(a, j+1, r) \\
            &QS(a, l, j-1)
        \end{align*}

end QS

Space $n$

Recursive cells stack

$QS(1, n) QS(2, n), QS(3, n), ..., QS(n-1, n)$
Iterative Quick Sort

QS(...,1,n)
   S <- NULL
   PUSH (S,(1,n))
   while S is not Empty do
      (l,r) <- POP(S)
      if l >= r go to end while
      ...
      if j-1-l <= r-(j+1)
         then Push(S,(j+1,r)) //large
            Push(S,(l,j-1)) //small
         else Push(S,(l,j-1)) //large
            Push(S,(j+1,r)) //small
      endwhile
   end QS
Iterative Quick Sort

Stack has length $O(\log n)$

Stack contents at different times for a problem with $\alpha$ values.

- $\alpha$
- $\alpha < \frac{\alpha}{2}$
- $\alpha < \frac{\alpha}{2} \leq \frac{\alpha}{4}$
- $\alpha < \frac{\alpha}{2} < \frac{\alpha}{4} \leq \frac{\alpha}{8}$
- ... 

Stack has length $O(\log n)$. 
Time Complexity

Worst Case

\[ T(n) \leq T(n - 1) + cn \]
\[ = T(n - 2) + c(n - 1) + cn \]
\[ = \ldots \]
\[ = T(1) + (c)2 + (c)3 + \ldots + c(n - 1) + c(n) \]
\[ = O(n^2) \]

There are examples that show that \( T(n) \) is \( \Omega(n^2) \).

Best Case
\[ T(n) = 2T(n/2) + cn \]
\[ \rightarrow O(n \log n) \], like merge sort but constant is smaller
Average Case

\[
T(n) \leq T(i) + T(n - i - 1) + cn
\]

\[
A(n) = \frac{2}{n} \sum_{j=0}^{n-1} (A(j)) + cn
\]

\[
nA(n) = 2 \sum_{j=0}^{n-1} A(j) + cn^2
\]

Subtracting from last equation
\[
(n - 1)A(n - 1) = 2 \sum_{j=0}^{n-2} A(j) + c(n - 1)^2
\]

we know that
\[
nA(n) - (n - 1)A(n - 1) = 2A(n - 1) + 2cn - c^\dagger
\]

\[
nA(n) = (n + 1)A(n - 1) + 2cn
\]

\[
\frac{A(n)}{n + 1} = \frac{A(n - 1)}{n} + \frac{2c}{n + 1}
\]

\[\dagger\]Note that \(-c\) is not significant!
Adding the following equations

\[
\frac{A(n)}{n + 1} = \frac{A(n - 1)}{n} + \frac{2c}{n + 1}
\]

\[
\frac{A(n - 1)}{n} = \frac{A(n - 2)}{n - 1} + \frac{2c}{n}
\]

\[
\frac{A(n - 2)}{n - 1} = \frac{A(n - 3)}{n - 2} + \frac{2c}{n - 1}
\]

\[
\vdots \quad \vdots \quad \vdots
\]

\[
\frac{A(2)}{3} = \frac{A(1)}{2} + \frac{2c}{3}
\]

We know that \( \frac{A(n)}{n + 1} = \frac{A(1)}{2} + 2c \sum_{j=3}^{n+1} \frac{1}{j} \)

The last summation is \( \log_e(n + 1) + 0.577 - \frac{3}{2} \)

Therefore, \( \frac{A(n)}{n + 1} = O(\log n) \)

Which is equivalent to \( A(n) = O(n \log n) \)
Randomized Quick Sort: Select the pivot at random with equal probability of selecting any of the elements being sorted.

Randomized Quick Sort has expected time complexity equal to $A(n)$, the average time complexity.

Randomized Quick Sort has worst case time complexity $O(n^2)$. But it is normally very fast.

Sort in Linux (Unix) uses Quick Sort.
Lower Bounds for Sorting

- **Decision Trees**: Binary tree
  - Nodes represent a set of possible orderings
  - A comparison is made between elements
  - The result is two set of possible orderings

Exp.

<table>
<thead>
<tr>
<th>a &lt; b &lt; c &lt; d</th>
<th>a &lt; c &lt; b &lt; d</th>
</tr>
</thead>
<tbody>
<tr>
<td>b &lt; c</td>
<td></td>
</tr>
</tbody>
</table>

T F

\[ a < b < c < d \quad a < c < b < d \]

Orderings
Assume elements are distinct

Every sorting method (that gains info only by comparing elements) may be viewed as a decision tree.
Lower Bound

- Sorting \( n \) elements \( x_1, x_2, x_3, \ldots, x_n \)
- \( n! \) leaves in any decision tree
- Height \( h \) of decision tree
  \[ h \geq \log_2 n! \]
  Since \( n! \geq \left( \frac{n}{2} \right)^{n/2} \)
  we know that \( h \) is \( \Omega(n \log n) \)

Sorting method that gain info only by comparing keys take time \( \Omega(n \log n) \)
The average number of comparison (for decision tree sorting) is equal to the Average Depth of a Leaf.

**Theorem:** For any decision tree with \( k \) leaves the \( ADL \geq \log_2 k \).

**Proof:** Proof by Contradiction

Let \( T \) be the least height decision tree with \( ADL < \log_2 k \), where \( k \) is the number of leaves in \( T \).
Lower Bound for Average TC: Cont’

- Let $k = k_1 + k_2$, then by Ind. Hypothesis
  $ADL(n_1) \geq \log_2 k_1, ADL(n_2) \geq \log_2 k_2,$
  $k = k_1 + k_2$

- $ADL \geq \frac{k_1}{k_1+k_2} \log_2 k_1 + \frac{k_2}{k_1+k_2} \log_2 k_2 + 1$

- $\rightarrow$ min when $k_1 = k_2 = k/2$

- $ADL$ is $\geq 0.5 \log k/2 + 0.5 \log k/2 + 1 = \log_2 k$

- A contradiction.

- Note that
  $(0.5 + \epsilon) \log(k/2 + \epsilon) + (0.5 - \epsilon) \log(k/2 - \epsilon) + 1$
  is greater than $\log_2 k$ simply because

- $\log(k/2 + \epsilon) - \log k/2 > \log k/2 - \log(k/2 - \epsilon)$
### Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>WC</th>
<th>AC</th>
<th>Stable?</th>
<th>Addl Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binsort*</td>
<td>$O(n)$</td>
<td></td>
<td></td>
<td>$O(n)$</td>
</tr>
<tr>
<td>RadixSort**</td>
<td>$O(kn)$</td>
<td></td>
<td></td>
<td>$O(n)$</td>
</tr>
<tr>
<td>HeapSort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>not stable</td>
<td>$O(c)$</td>
</tr>
<tr>
<td>QuickSort</td>
<td>$O(n^2)$</td>
<td>$O(n \log n)$</td>
<td>not stable</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Mergesort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>stable</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Insertsort***</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>stable</td>
<td></td>
</tr>
</tbody>
</table>

Binsort*: Sorts integers in the range $[0, n - 1]$. RadixSort**: Sorts integers in the range $[0, n^k - 1]$. Insertsort***: Best method when sorting $\leq 15$ elements, used in hybrid algorithms.

• $O(n \log \log n)$ time and $O(n)$ space, “Not practical”
  Y. Han, Determinist Sorting in $O(n \log \log n)$ time + Linear Space, STOC’02, pp. 602-608. Sorts Integers Only.