Common Sorting Algorithms [1]

Donald Dong

Monterey Bay Programming Team

xdong@csumb.edu

February 14, 2018

Overview

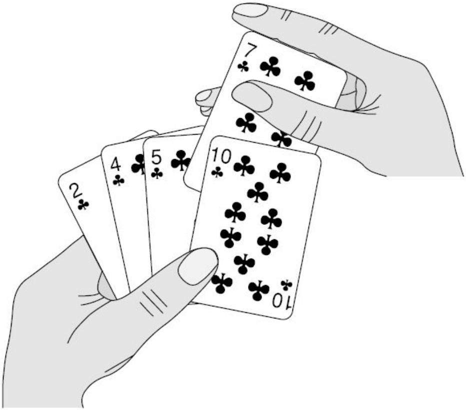
Why Sorting?

- Fast Search
  - Binary Search
  - Exponential Search
- Algorithms often use sorting as a key subroutine
  - Uniqueness
  - Palindrome
  - Events Scheduling

Characteristics

- Running Time: The number of primitive operations or steps executed
  - Worst-case
  - Average-case
  - Best-case
- In-place: The output is placed in the correct position while the algorithm is still executing
- Stable: Two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array

Insertion Sort



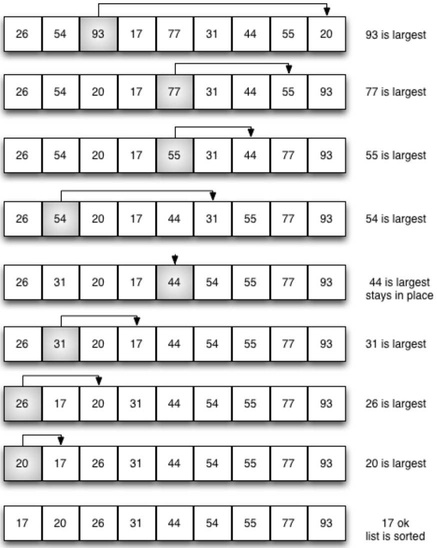
How to sort a hand of cards using insertion sort?

Insertion Sort

Algorithm Insertion Sort($A[0..n-1]$)
Input: Array $A[0..n-1]$ of orderable values
Output: Array $A[0..n-1]$ in sorted in non-decreasing order

for $i \leftarrow 1$ **to** $n-1$ **do**
    $key \leftarrow A[i]$
    $j \leftarrow i-1$
    while $j \geq 0$ and $A[j] > key$ **do**
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j-1$
    $A[j+1] \leftarrow key$

Selection Sort

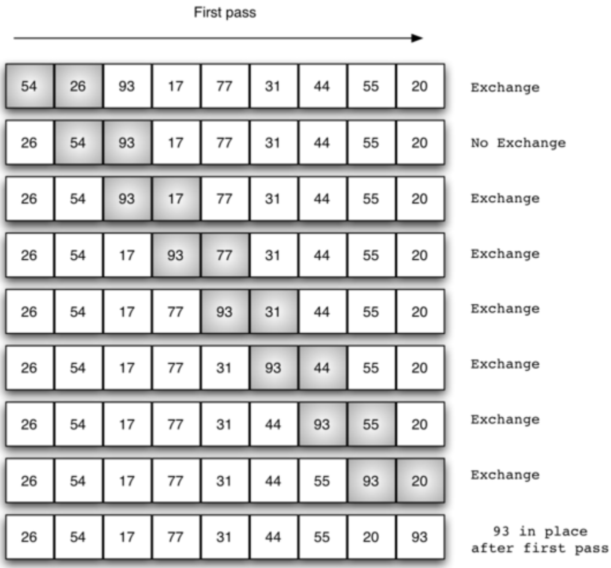


Selection Sort

Algorithm Selection Sort($A[0..n-1]$)
Input: Array $A[0..n-1]$ of orderable values
Output: Array $A[0..n-1]$ in sorted in non-decreasing order

```
for i ← 0 to n − 2 do
    min ← i
    for j ← i + 1 to n − 1 do
        if A[j] < A[min] then
            min ← j
    swap(A[i], A[min])
```

Bubble Sort

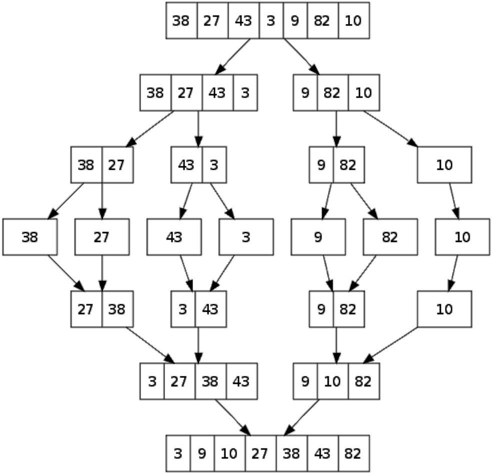

First pass
54 26 93 17 77 31 44 55 20
Exchange
26 54 93 17 77 31 44 55 20
No Exchange
26 54 93 17 77 31 44 55 20
Exchange
26 54 17 93 77 31 44 55 20
Exchange
26 54 17 77 93 31 44 55 20
Exchange
26 54 17 77 31 93 44 55 20
Exchange
26 54 17 77 31 44 93 55 20
Exchange
26 54 17 77 31 44 55 93 20
Exchange
26 54 17 77 31 44 55 20 93
93 in place
after first pass

Bubble Sort

Algorithm Bubble Sort($A[0..n-1]$)
Input: Array $A[0..n-1]$ of orderable values
Output: Array $A[0..n-1]$ in sorted in non-decreasing order

for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow 0$ **to** $n-2-i$ **do**
        if $A[j+1] < A[j]$ **then**
            $swap(A[j], A[j+1])$

Merge Sort



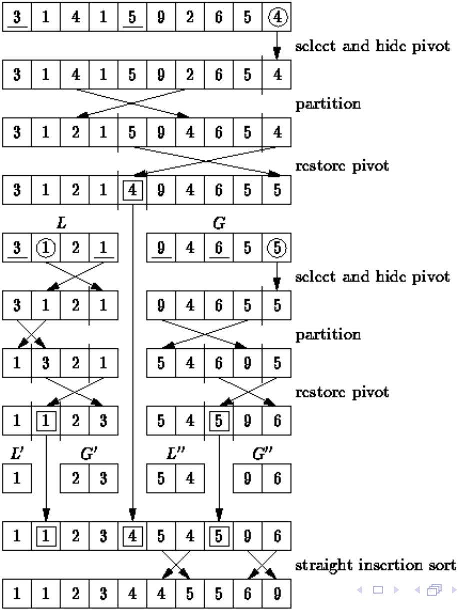
38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Merge Sort

```
MERGE(A, p, q, r)
l ← q − p + 1
r ← r − q
Let L[0..l] and R[0..r] be new arrays
for i ← 0 to l − 1 do
    L[i] ← A[p + i]
for j ← 0 to r − 1 do
    R[j] ← A[q + 1 + j]
L[l] ← ∞
R[r] ← ∞
i ← 0
j ← 0
for k ← p to r do
    if L[i] ≤ R[j] then
        A[k] ← L[i]
        i ← i + 1
    else
        A[k] ← R[j]
        j ← j + 1
```

```
MERGE-SORT(A, p, r)
if p < r then
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Quick Sort

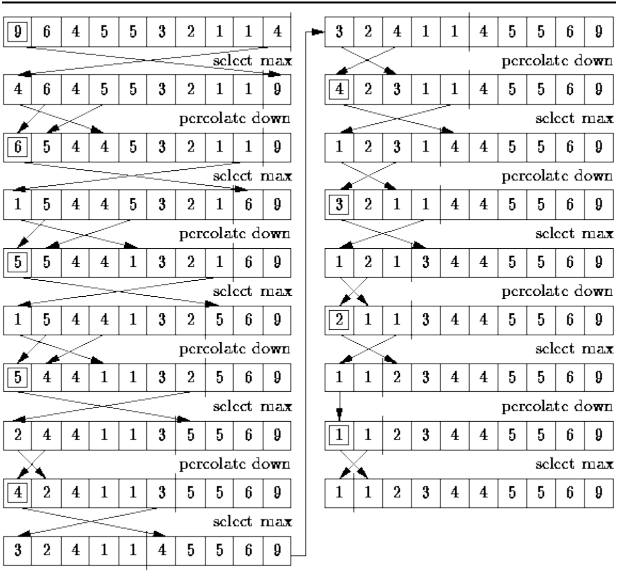

select and hide pivot
partition
restore pivot
L
G
select and hide pivot
partition
restore pivot
L'
G'
L''
G''
straight insertion sort

Quick Sort

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] < x then
        i ← i + 1
        SWAP(A[i], A[j])
SWAP(A[i + 1], A[r])
return i + 1
```

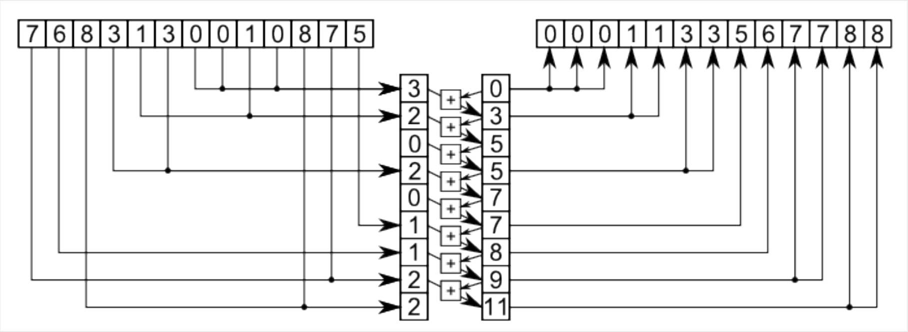
```
QUICK-SORT(A, p, r)
if p < r then
    q = PARTITION(A, p, r)
    QUICK-SORT(A, p, q − 1)
    QUICK-SORT(A, p + 1, q)
```

Heap Sort

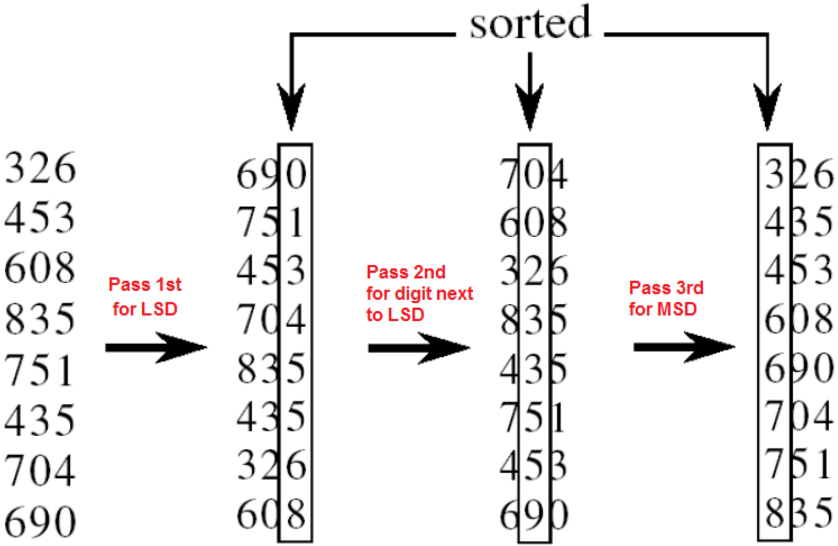


9 6 4 5 5 3 2 1 1 4
select max
4 6 4 5 5 3 2 1 1 9
percolate down
6 5 4 4 5 3 2 1 1 9
select max
1 5 4 4 5 3 2 1 6 9
percolate down
5 5 4 4 1 3 2 1 6 9
select max
1 5 4 4 1 3 2 5 6 9
percolate down
5 4 4 1 1 3 2 5 6 9
select max
2 4 4 1 1 3 5 5 6 9
percolate down
4 2 4 1 1 3 5 5 6 9
select max
3 2 4 1 1 4 5 5 6 9
3 2 4 1 1 4 5 5 6 9
percolate down
4 2 3 1 1 4 5 5 6 9
select max
1 2 3 1 4 4 5 5 6 9
percolate down
3 2 1 1 4 4 5 5 6 9
select max
1 2 1 3 4 4 5 5 6 9
percolate down
2 1 1 3 4 4 5 5 6 9
select max
1 1 2 3 4 4 5 5 6 9
percolate down
1 1 2 3 4 4 5 5 6 9
select max
1 1 2 3 4 4 5 5 6 9

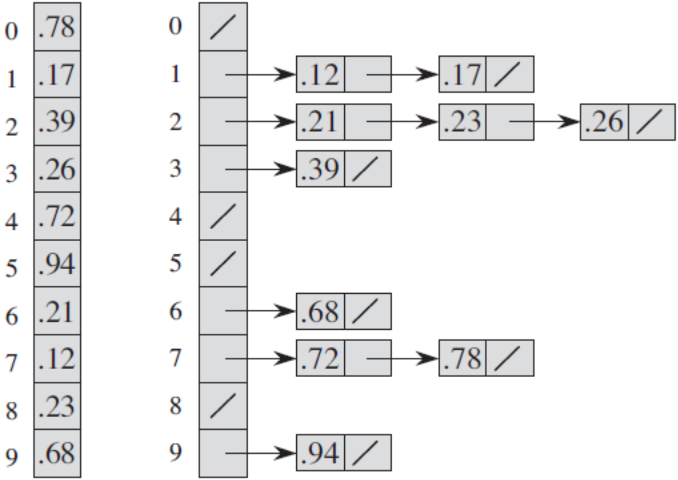
Counting Sort



Radix Sort


sorted
326
453
608
835
751
435
704
690
Pass 1st
for LSD
690
751
453
704
835
435
326
608
Pass 2nd
for digit next
to LSD
704
608
326
835
435
751
453
690
Pass 3rd
for MSD
326
435
453
608
690
704
751
835

Bucket Sort



Input array

Buckets created for input array

Comparison Counting Sort

Algorithm Comparison Counting Sort($A[0..n-1]$, $S[0..n-1]$)
Input: Array $A[0..n-1]$ of orderable values
Output: Array $S[0..n-1]$ of A's elements sorted in non-decreasing order

```
for i ← 0 to n − 1 do
    Count[i] ← 0
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i] < A[j] then
            Count[j] ← Count[j] + 1
        else
            Count[i] ← Count[i] + 1
for i ← 0 to n − 1 do
    S[Count[i]] ← A[i]
```

Problem A

Suppose that there is a restaurant and we know the arriving and leaving times of all customers on a certain day. Our task is to find out the maximum number of customers who visited the restaurant at the same time.

Input: Arriving & Leaving

1 6

3 5

2 8

Output: Number of customers

3

Problem A - 2

Suppose that there is a restaurant and we know the arriving and leaving times of all customers on a certain day. Our task is to find out the maximum number of customers who visited the restaurant in certain time interval.

Input: Arriving & Leaving
3
1 6
3 5
2 8
2
1 2
1 8

Output: Number of customers
2
3

Problem B

Given n events with their starting and ending times, find a schedule that includes as many events as possible.

Input: Starting & Ending

7

9 14

23 32

0 15

17 29

26 32

13 22

3 12

Output: Number of events

3

References

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
Introduction to Algorithms, Third Edition.
The MIT Press, 3rd edition, 2009.