Topic 3 - Sorting

Read the CLRS chapter on sorting  
What is sorting? Rearrange into a specific order.

How many orders are there?

What is the opposite of sorting?

n=8

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 1 | 2 | 3 | 6 | 5 | 4 |

n!

|  |
| --- |
| 1 |

1 order

|  |  |
| --- | --- |
| 2 | 1 |

2 orders 2!

n! if the elements are distinct

The opposite of sorting. Start with a sorted list

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

shuffling (from cards)

pick a random disordered list with the same values

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 7 | 1 | 6 | 7 | 4 |

Why is it easier to shuffle than sort: entropy

Sorting Algorithm

Bubble (bad)

Selection (bad)

Insertion

Quicksort

Heapsort

Mergesort

# Bubble sort

Example: sort into ascending order

each comparison O(1) each swap O(1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 1 | 2 | 3 | 6 | 5 | 4 |
| 7 | 8 |  |  |  |  |  |  |
|  | 1 | 8 |  |  |  |  |  |
|  |  | 2 | 8 |  |  |  |  |
|  |  |  | 3 | 8 |  |  |  |
|  |  |  |  | 6 | 8 |  |  |
|  |  |  |  |  | 5 | 8 |  |
|  |  |  |  |  |  | 4 | 8 |
| 7 | 1 | 2 | 3 | 6 | 5 | 4 | 8 |

(n-1) + (n-2) + … (1)

=O(n2)

bubblesort(x)

for (int i = 1; i < x.length; i++)

for (int j = 0; j < x.length-i-1; j++)

if (x[j] > x[j+1])

temp ← x[j]

x[j] ← x[j+1]

x[j+1] ← temp

end

end

end

end

swap (a, b)

temp ← x[j]

x[j] ← x[j+1]

x[j+1] ← temp

end

swap(a, b)

x[j] ← x[j] + x[j+1]

x[j+1] ← x[j] - x[j+1]

x[j] ← x[j] - x[j+1]

end

swap(a, b)

x[j] ← x[j] ^ x[j+1]

x[j+1] ← x[j] ^ x[j+1]

x[j] ← x[j] ^ x[j+1]

end

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

O(n2) Ω(n)

bubblesortTerminateEarly(x)

for (int i = 1; i < x.length; i++)

swapped ← false

for (int j = 0; j < x.length-i-1; j++)

if (x[j] > x[j+1])

temp ← x[j]

x[j] ← x[j+1]

x[j+1] ← temp

swapped ← true

end

end

if NOT swapped

quit

end

end

end

# Selection Sort

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 8 | 5 | 6 | 7 | 1 |
|  |  |  | 1 |  |  |  | 8 |

O(n2) Ω(n2) ⇒ Θ(n2)

selectionsort(x)

for (int j = x.length - 1; j > 0; j--) {

int max = x[0];

int pos =0;

for (int i = 1; i <= j; i++)

if (x[i] > max) {

max = x[i];

pos = i;

}

temp = x[pos];

x[pos] = x[j];

x[j] = temp;

}

# Insertion Sort

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 7 | 1 | 6 | 7 | 4 |
| 2 | 8 |  |  |  |  |  |  |
| 2 | 3 | 8 |  |  |  |  |  |
| 2 | 3 | 7 | 8 |  |  |  |  |
| 1 | 2 | 3 | 7 | 8 |  |  |  |
|  |  |  |  |  |  |  |  |

1 + 2 + 3 + …. + (n-1) = O(n2)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  |  |  |

1 + 1 + 1 + … + 1 = Ω(n)

# 

# 

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 4 | 5 | 6 | 7 | 1 |

1, 2, 3, 4, 5, 6, …, 52

52, 1, 6, 51, … // O(n2)

sorting a deck of cards (1..52) = O(n)! Just rewrite

for i = 1; i <= 52; i++)

x[i] = i;

# 

# 

# Quicksort

To improve beyond O(n2) we need algorithms that rapidly bring ***multiple elements*** closer to their correct position

First, find a value in the middle (pivot)

1. pivot = (x[L] + x[R]) /2 = (8 + 4)/2 = 6
2. pivot = x[random] = x[3] = 7 random is unpredictable!
3. pivot = (x[L] + x[R] + x[(L+R)/2]) / 3 = (8 + 4 + x[3])/3 = (8 + 4 + 7) / 3 = 19/3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 7 | 1 | 6 | 7 | 4 |
| L |  |  |  |  |  |  | R |

x[i] < pivot x[i] == pivot x[i] > pivot

x[i] < pivot x[i] >= pivot

pivot = 6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 7 | 1 | 6 | 7 | 4 |
| L |  |  |  |  |  |  |  |
| i |  |  |  |  |  |  | j |
| 4 | i | i | i |  | j | j | 8 |
| 4 | 2 | 3 | 6 | 1 | 7 | 7 | 8 |
|  |  |  |  | ij |  |  |  |

quicksort(x, L, R)

if R <= L

return

end

i← L

j← R

while i < j

while x[i] <= pivot

i← i + 1

end

while x[j] > pivot

j ← j-1

end

if i < j

swap(x[i], x[j])

quicksort(x, L, i)

quicksort(x, i+1, R)

end

quicksort(x, 0, N-1) // O(n)

quicksort(x, 0, i) // O(n)

quicksort(x, i+1, N-1)

quicksort( ) //O(n)

quicksort( )

quicksort( )

quicksort( )

O(n log n)

pivot = x[L]

pivot = x[R]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

quicksort(x, 0, 6)

quicksort(x, 7,7)

pivot = x[(L+R)/2]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

pathological worst case!

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 5 | 1 | 6 | 3 | 7 | 8 |

pivot = (x[L] + x[R])/2

pathological worst case!

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 5 | 7 | 8 | 6 | 4 | 2 |

|  |  |  |
| --- | --- | --- |
| n | n2 | n log2 n |
| 101 | 100 | 101\*3 |
| 103 | 106 | 103\*10 |
| 106 | 1012 | 106\*20 |
| 109 | 1018 | 109\*30 |

Use the random element pivot!

No matter what, it always works

## Knuth Optimized Quicksort

n = 106

quicksort(x, 0, n-1) // O(n)

quicksort()quicksort() //O(n)

…

quicksort( ) …. 500,000 //O(n) but VERY HIGH constant

Knuth optimization: Sort until no partition > k elements

then use insertion sort

quicksortOpt(x, L, R)

…

if i - L > k

quicksortOpt(x, L, i)

end

if R - (i+1) > k

quicksortOpt(x, i+1, R)

end

quicksortOptimized(x)

quicksortOpt(x, 0, x.length-1)

insertionsort(x)

end

# 

# 

# Heapsort

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

leftchild(i) = 2i+1, rightchild(i) = 2i + 2



for i = n/2; i >= 0; i-- // O(n)

inserting into tree O(log n)

makeheap O(n log n)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **8** | **5** | **7** | **2** | **1** | **6** | **3** | **4** |
| **4** |  |  |  |  |  |  | **8** |

Fixing the tree is O(log n)

Makeheap = O(n log n) + log n + log n …..

O(n log n ) + O(n log n) = O(n log n)

# 

# Merge Sort

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 7 | 1 | 6 | 7 | 4 |

<http://image1.masterfile.com/getImage/846-05647748em-1960s-MAN-TECHNICIAN-PROGRAMMING-MAINFRAME-COMPUTER-AT-CONTROL-ROOM-CONSOLE-SURROUNDED-BY-MANY-TAPE-DRIVE-DATA-STORAGE.jpg>

Video of a tape drive

<https://www.youtube.com/watch?v=Nq3mNYKR7FM>

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 7 | 1 | 6 | 7 | 4 |
| 2 | 8 | 3 | 7 | 1 | 6 | 4 | 7 |
| i | i | j | j |  |  |  |  |
| 2 | 3 | 7 | 8 | 1 | 4 | 6 | 7 |
|  |  |  |  |  |  |  |  |

A

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 7 | 1 | 6 | 7 | 4 |

B

n additional units of storage (reuse the array)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 8 | 3 | 7 | 1 | 6 | 4 | 7 |

reuse A

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 2 | 3 | 7 | 1 | 6 | 7 | 4 |

Radix Sort (not required)

000000001010101010010101010010111110000101010101

n=109

1234 9871 4936 1277 ….

n1= 1.2341 x 108 numbers starting with 1

devise a number of bins

List leadingDigits[10];

for each v in list

leadingDigits[v / 1000].add(v)

deck ← 1..52

shuffle (deck is random)

sort(deck)

for i← 0 to 51

deck[i] ← i+1