More sorts

Insertion sort

Imagine we have alist of numbers from 1-10^10

[…….7,2] random sorted list

Insertion sort: each iteration, the k leftmost indies are all relatively sorted

With 2 iteratiions remainng, [1,3,4,5,6,8,9,10……999999,7,2]

7 has to march down to find the correct spot

2 also has to march a long way, do a lot of comparisons and get to its final spot

Would help if 7 and 2 were compared and relatively sorted together first

Now has pairs or better with grouping and then sorting relatively

In the limit, place half the data relatively sorted into the other half

\*optimal limit is basically up to where you have 2 halves of sorted/grouped data

Merge sort\*\* ^ - first example of a divide and conquer algorithm – recursively done

Method: Split data into 2 halves, recursively call sorting algorithm, after sorting, merge 2 halves back together

\*\* pseudocode \*\*

Def mergesort(lst):

If(len(lst) <= 1:

Return L #sorted and claim its sorted so just return it after basecase is done, returning a #new list

Else:

(half1,half2) = split(lst)

Return (merge(mergesort(half1), mergesort(half2))

#we write functions for merge, split

Example: \*\*

[5,7,1,3,4,8,6,2]

#first split data into 2 halves

Could split just list in half, but also by evns, odds indices

[5,1,4,6] even index [7,3,8,2] odd index

[5,4] [1,6] [7,8] [3,2]

[5] [4] [1] [6] [7] [8] [3] [2]

[4,5] [1,6] [7,8] [2,3] binary tree-like structure

[1,4,5,6] [2,3,7,8]

[1,2,3,4,5,6,7,8] done, sorted

[…….] list 1 sorted

[…….] list 1 + 2 merge sorted

[…….] list 2 sorted

2 lists that are sorted, keep a pointer for larger or smaller depending on values, and compare from there, moving only if one list has a larger value

Any leftovers in the smaller list overall go into the merged list at the end

Complexity\*\*\*

Split(n items) – d(n) – constant out front, have to touch each element,

Merge(two lists of size n/2) - d(n)

Log(n) operation on the depth of the tree complexity

The d is the amount of work needed to operate on those lists, returns the constant

Have list of size n,

0. Work d(n)

1. d(n/2)2 = d(n)

2. d(n/4)4 = d(n)

….. merge everything back

3. at each level of the tree and merge it back together, is d(n)

Total to break everything down is log(n) of everything down, o(n)

Total overall complexity is o(n\*log(n))

Stress.py

From random import \*

From time import \*

From insertionSort import insertionSort

From mergeSort import MergeSort

From quicksort import quicksort

From copy import deepcopy

Listsize=10000

#deep copy ensure new list

\*\* break \*\*

QuckSort:

Another divide and conquer algorithm

Expected complexity is o(n\*log(n))

Worst case is o(n^2)

Whereas mergeSort backloaded its work, split is easy, merge a little more work

Picks a pivot and has 3 “buckets”

Equal to, greater than, less than

Recursively with less, greater, then concatenate back to the equal ones

\*\*

Quicksort does more work up front

Partitions 3 buckets (less, equal, more)

Just concatenate the pieces

Pivot: often just the first value in the list

Partition: recursively call quicksort on less and greater than sub-lists

Kind of similar to quick select sorting method

But don’t throw away one part, keep both and sort everything

\*\*pseudocode\*\*

Def quicksort(lst):

If(lst==[]):

Return []

Else:

(less,same,more) = partition(lst) #equal may be a keyword in python

Return quicksort(less) + same + quicksort(more)

#have to write partition, pick pivot and put the stuff in it

[5 7 1 3 4 8 6 2]

[1 3 4 2] [5] [7 8 6] # 5 is pivot

[ ] [1] [3 4 2] #5 carries down [6] [7] [8]

[ ] [1] [2] [3] [4]

#each is individually given empty lists and then rejoined once they are single lists

[ ] [1 2 3 4] [5] [6 7 8]

[1 2 3 4 5 6 7 8]

\*\*complexity\*\*

Best case:

#break into even pieces, balanced, complete binary tree

Very similar to mergesort for this

Pivot evenly splits data for o(n\*log(n))

Worst case: #already sorted list

Pivot is largest or smallest element #one bucket is empty other has n-1 elements in it (account for pivot)

D(n) = n+t(n-1)

+ (n-1) + t(n-2)

N+ (n-1) + (n-2) + …..1 = n(n^2)

Def partition(pivot lst):

(less,same,more) = ([], [], [])

For e in lst:

If e< pivot:

Less.append(e)

Elif e>pivot:

More.append(e)

Else:

Same.append(e)

Return (less, same, more)