CMSC201
Computer Science I for Majors
Lecture 24 – Sorting

Prof. Jeremy Dixon

Based on slides from previous iterations of the course
Surveys

• Blackboard Survey – worth 1% of your final grade. Take a few minutes to complete it.
  – Due Sunday, December 6th

• SCEQ – provides feedback about instructors and the course itself

• BRAID Survey – December/January version
  – You should receive an email link
Last Class We Covered

• Searching
  – Linear search
  – Binary search

• Asymptotic Performance
  – How fast an algorithm “runs”
  – Why certain algorithms are “better” than others
Any Questions from Last Time?
Today’s Objectives

• To learn about sorting algorithms
  – Selection Sort
  – Bubble Sort
  – Quick Sort
  – Radix Sort

• To examine which of these algorithms is best for different sorting situations

• Surveys
Sorting
Sorting Algorithms

• Sorting algorithms put the elements of a list in a specific order

• A sorted list is necessary to be able to use certain other algorithms

• Like binary search!
  – If sorted once, we can search many, many times
Sorting Algorithms

• There are many different ways to sort a list
• What method would you use?
• Now imagine you can only look at at most two elements at a time

• Computer science has a number of commonly used sorting algorithms
Selection Sort
Selection Sort Algorithm

• Here is a simple way of sorting a list:

1. Find the smallest number in a list
2. Move that to the end of a new list
3. Repeat until the original list is empty
Selection Sort Run Time

• What is the Big Oh of finding the lowest number in a list?

• For a list of size $N$, what is the worst case number of elements you’d have to look through to find the min?

• $N$
Selection Sort Run Time

• For a list of size $N$, how many times would we have to find the min to sort the list?
  • $N$

• What is the Big Oh of this sorting algorithm?
  • $O(N^2)$
Bubble Sort
Bubble Sort Algorithm

• Let’s take a look at another sorting method!

1. We look at the first pair of items in the list, and if the first one is bigger than the second one, we swap them

2. Then we look at the second and third one and put them in order, and so on

3. Once we hit the end of the list, we start over at the beginning

4. Repeat until the list is sorted!
Bubble Sort Example

[ 4, 8, 1, 10, 13, 14, 6]

First pass:
4 and 8 are in order
8 and 1 should be swapped:
[ 4, 1, 8, 10, 13, 14, 6]

8 and 10 are in order
10 and 13 are in order
13 and 14 are in order
6 and 14 should be swapped:
[ 4, 1, 8, 10, 13, 6, 14]
Bubble Sort Example (Cont)

[ 4, 1, 8, 10, 13, 6, 14]

Second pass:
4 and 1 should be swapped:
[ 1, 4, 8, 10, 13, 6, 14]

4 and 8 are in order
8 and 10 are in order
10 and 13 are in order
13 and 6 should be swapped:
[ 1, 4, 8, 10, 6, 13, 14]
13 and 14 are in order
Bubble Sort Example (Cont)

[ 4, 1, 8, 10, 13, 6, 14]

Second pass:
4 and 1 should be swapped:
[ 1, 4, 8, 10, 13, 6, 14]

4 and 8 are in order
8 and 10 are in order
10 and 13 are in order
13 and 6 should be swapped:
[ 1, 4, 8, 10, 6, 13, 14]
13 and 14 are in order
Bubble Sort Example (Cont)

[ 1, 4, 8, 10, 6, 13, 14]

Third pass:
10 and 6 should be swapped:
[ 1, 4, 8, 6, 10, 13, 14]

Fourth pass:
8 and 6 should be swapped:
[ 1, 4, 6, 8, 10, 13, 14]
Bubble Sort Run Time

• For a list of size $N$, how much work do we do for a single pass?
  - $N$

• How many passes will we have to do?
  - $N$

• What is the Big Oh of Bubble Sort?
  - $O(N^2)$
Bubble Sort Code

def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)
Quicksort
Quicksort Algorithm

• Here’s another method:
1. Start with the number on the far right
2. Put everything less than that number on the left of it and everything greater than it on the right of it
3. Quicksort the left side and the right side

• Does this method remind you of anything?
Quicksort Run Time

• For a list of size \( N \), how many steps does it take to move everything less than the last number to the left and everything greater than the last number to the right?

• \( N \)
Quicksort Run Time

- How many times does the algorithm divide the list in half?
  - $\lg(N)$

- What is the Big Oh of Quicksort?
  - $O(N \ lg(N))$
def quickSort(alist):
    quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
    if first<last:
        splitpoint = partition(alist,first,last)
        quickSortHelper(alist,first,splitpoint-1)
        quickSortHelper(alist,splitpoint+1,last)

def partition(alist,first,last):
    pivotvalue = alist[first]
    leftmark = first+1
    rightmark = last
    done = False
    while not done:
        while leftmark <= rightmark and alist[leftmark] <= pivotvalue:
            leftmark = leftmark + 1
        while alist[rightmark] >= pivotvalue and rightmark >= leftmark:
            rightmark = rightmark -1
        if rightmark < leftmark:
            done = True
        else:
            temp = alist[leftmark]
            alist[leftmark] = alist[rightmark]
            alist[rightmark] = temp

            temp = alist[first]
            alist[first] = alist[rightmark]
            alist[rightmark] = temp

    return rightmark

alist = [54,26,93,17,77,31,44,55,20]
quickSort(alist)
print(alist)
Radix Sort
Improving Run Time

• Most of the time, $O(N \log(N))$ is the best we can do for sorting

• However if we make the problem slightly easier, we can do even better!

• Imagine we know for a fact that the list we are sorting is only integers between 0 and 9
Radix Sort Algorithm

• We can make a list of size 10 filled with zeroes
• The first element of this list represents the number of zeroes we’ve seen so far in the list we’re sorting
• The second number is the number of ones we’ve seen, and so on
Radix Sort Algorithm

• So say we have the list:
  – [0, 3, 2, 1, 6, 8]

• We make our counting list:
  – [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

• And iterate over the list we want to sort
Radix Sort Algorithm

• The first number is a zero, so we add one to the zeroth element of our counting list:
  – [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

• The next number is a 3, so we add one to the third element of our counting list:
  – [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Radix Sort Algorithm

• Then 2:
  – [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]

• Then 1:
  – [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

• When we’re done, the list looks like this:
  – [1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0]
Radix Sort Algorithm

• For an index \(i\), we know that if \(\text{countList}[i] == 1\), there was one \(i\) in the original list

• One pass over the counting list to figure out which numbers were there and we’ve sorted it!
Radix Sort Run Time

• How many operations do we need to do to fill out our counting list with zeros?
  – $N$

• How many operations do we need to do to fill out our counting list with the right values?
  – $N$
Radix Sort Run Time

• How many operations do we need to do to reconstruct our sorted list?
  – $N$

• This gives us a total run time of $3N$ operations
  – So our final run time is simply
  – $O(N)$
def radixsort( aList ):
    RADIX = 10
    maxLength = False
tmp , placement = -1, 1

while not maxLength:
    maxLength = True
# declare and initialize buckets
buckets = [list() for _ in range( RADIX )]

# split aList between lists
for i in aList:
    tmp = i / placement
    buckets[tmp % RADIX].append( i )
    if maxLength and tmp > 0:
        maxLength = False

# empty lists into aList array
a = 0
for b in range( RADIX ):
    buck = buckets[b]
    for i in buck:
        aList[a] = i
        a += 1

# move to next digit
placement *= RADIX
Python Built-in `sort()`

- Python’s built-in sorting method `sort()` uses a hybrid sort called a timsort which was invented by Tim Peters in 2002 for use with Python

- [http://bugs.python.org/file4451/timsort.txt](http://bugs.python.org/file4451/timsort.txt)
Any Other Questions?
General Announcements

• Lab 12 this week – last lab of the semester!

• Project 2 is out
  – Due by Tuesday, December 8th at 8:59:59 PM
  – Do NOT procrastinate!

• Next Class: Review for the Final
Announcements: Final Exam

• Final Exam will be held on Friday, December 11th from 3:30 to 5:30 PM
• Being held in three separate rooms
  • Section 1 (Gibson, MW @ 1) – CHEM 030
  • Section 7 (Dixon, TR @ 5:30) – CHEM 030
  • Section 13 (Dixon, TR @ 10) – CHEM 030
  • Section 19 (Morawski, MW @ 4) – PAHB 132
  • Section 25 (Gibson, TR @ 4) – PHYS 101
• Make sure you go to the correct room!
Announcements: Surveys

• The second survey will be released and announced on Blackboard shortly
  – This is 1% of your grade, and is your chance to give feedback on your experience with the course

• Now, we will be doing the in-class SCEQ (Student Course Evaluation Questionnaire)
  – This is an important metric for assessment
SCEQ Details

• Use only a #2 pencil
• Catalog number should be in top left corner
• Fill in the number of credits earned towards your degree at the beginning of the semester
  – If less than 100, fill the two right-most columns
  – If less than 10, fill the right-most column
• Fill in your cumulative GPA
  – Fill unknown digits with “0”
SCEQ Details

• Fill in your officially declared major

  Computer Sci  63  Applied Physics  62
  Computer Eng  07  Atmo Physics  41
  Information Sys  83  Eng (General)  76
  Math  61  Chemical Eng  37
  Bioinformatics  98  Biology  55

  – If you haven’t declared a major, enter “00”
  – If yours isn’t listed, raise your hand and I’ll tell you
SCEQ Details

• For this course, fill out the Scantron (using a pencil), sections:
  – A (General)
  – B (Lecture) – “Instructor A” column only
  – D (Laboratory)

• Fill out the Blue sheet
  – Additional comments can be written on the back

• Bring completed sheets to the front