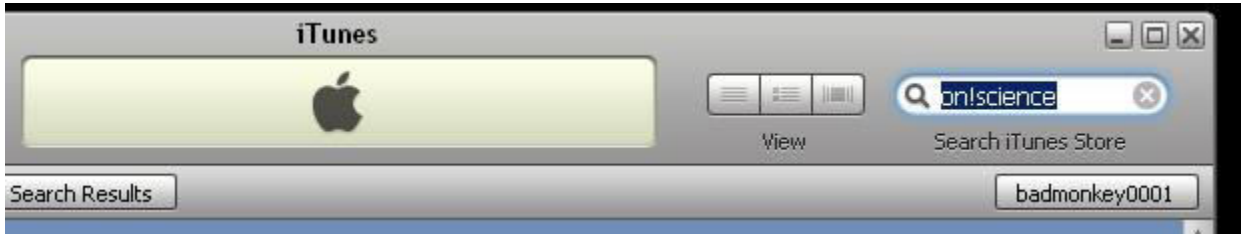
Sorting and Searching

Sorting and Searching

- Fundamental problems in computer science and programming
- Sorting done to make searching easier
- Multiple different algorithms to solve the same problem
  - How do we know which algorithm is "better"?
- Look at searching first
- Examples will use arrays of ints to illustrate algorithms

Searching


iTunes
bn!science
View
Search iTunes Store
Search Results
badmonkey0001


Google™
recursive backtracking
Google Search
I'm Feeling Lucky
Advanced Search
Preferences
Language Tools


Grep in Project
Grep in Project
/Library/Application Support/TextMate/Support
Searching for "automatic"
11 matching lines:
./bin/CocoaDialog-license.txt:175 void, and will automatically terminate your rights under this License.
./bin/CocoaDialog-license.txt:190 Program), the recipient automatically receives a license from the
Binary file ./bin/CocoaDialog.app/Contents/MacOS/CocoaDialog
./bin/CocoaDialog.app/Contents/Resources/COPYING:175 void, and will automatically terminate your rights under this License.
./bin/CocoaDialog.app/Contents/Resources/COPYING:190 Program), the recipient automatically receives a license from the
./bin/SmartyPants-license.txt:93 filter automatically applies SmartyPants to the bodies of your entries;
./bin/SmartyPants-license.txt:101 automatically. Textile i ··· web text generator",
./bin/SmartyPants-license.txt:166 3. That's it. The entries in your weblog should now automatically have
./bin/SmartyPants-license.txt:572 * Added a new option to automatically convert `&quot;` entities into
./bin/SmartyPants.pl:1070 + Added a new option to automatically convert " entities into
./lib/Builder.rb:282 # > and & automatically. Use the <tt><<</tt> operation to

Searching

- Given a list of data find the location of a particular value or report that value is not present
- linear search
  - intuitive approach
  - start at first item
  - is it the one I am looking for?
  - if not go to next item
  - repeat until found or all items checked
- If items not sorted or unsortable this approach is necessary

Linear Search

```
/*     pre: list != null
       post: return the index of the first occurrence
       of target in list or -1 if target not present in
       list
*/
public int linearSearch(int[] list, int target) {
    for(int i = 0; i < list.length; i++)
        if( list[i] == target )
            return i;
    return -1;
}
```

Linear Search, Generic

```
/*     pre: list != null, target != null
       post: return the index of the first occurrence
       of target in list or -1 if target not present in
       list
*/
public int linearSearch(Object[] list, Object target) {
    for(int i = 0; i < list.length; i++)
        if( list[i] != null && list[i].equals(target) )
            return i;
    return -1;
}
```

T(N)? Big O? Best case, worst case, average case?

Attendance Question 1

- What is the average case Big O of linear search in an array with N items, if an item is present?

A. O(N)

B. $O(N^2)$

C. O(1)

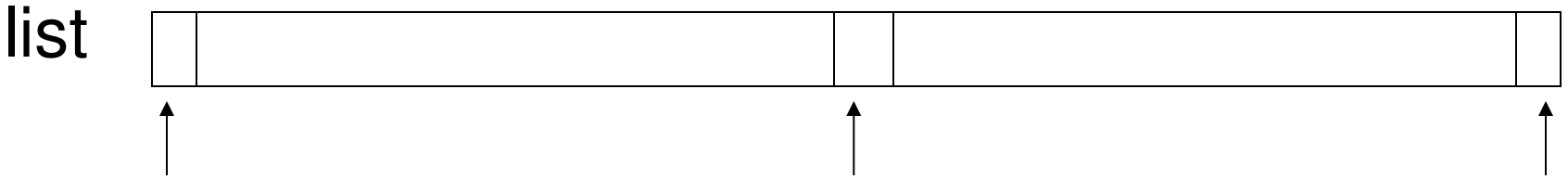
D. O(logN)

E. O(NlogN)

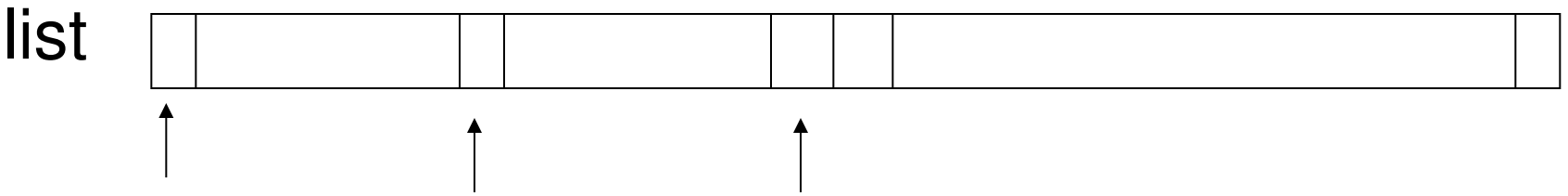
Searching in a Sorted List

- If items are sorted then we can *divide and conquer*
- dividing your work in half with each step
  - generally a good thing
- The Binary Search on List in Ascending order
  - Start at middle of list
  - is that the item?
  - If not is it less than or greater than the item?
  - less than, move to second half of list
  - greater than, move to first half of list
  - repeat until found or sub list size = 0

Binary Search



Is middle item what we are looking for? If not is it more or less than the target item? (Assume lower)



and so forth…

Binary Search in Action

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53

```
public static int bsearch(int[] list, int target)
{       int result = -1;
        int low = 0;
        int high = list.length - 1;
        int mid;
        while( result == -1 && low <= high )
        {       mid = low + ((high - low) / 2);
                if( list[mid] == target )
                        result = mid;
                else if( list[mid] < target)
                        low = mid + 1;
                else
                        high = mid - 1;
        }
        return result;
}
// mid = ( low + high ) / 2; // may overflow!!!
// or mid = (low + high) >>> 1; using bitwise op
```

Trace When Key == 3

Trace When Key == 30

Variables of Interest?

Attendance Question 2

What is the worst case Big O of binary search in an array with N items, if an item is present?

A. O(N)

B. $O(N^2)$

C. O(1)

D. O(logN)

E. O(NlogN)

Generic Binary Search

```
public static int bsearch(Comparable[] list, Comparable target)
{       int result = -1;
        int low = 0;
        int high = list.length - 1;
        int mid;
        while( result == -1 && low <= high )
        {       mid = low + ((high - low) / 2);
                if( target.equals(list[mid]) )
                        result = mid;
                else if(target.compareTo(list[mid]) > 0)
                        low = mid + 1;
                else
                        high = mid - 1;
        }
        return result;
}
```

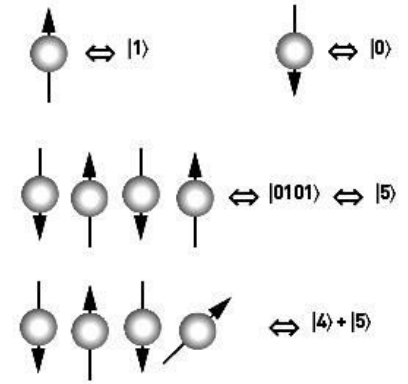
Recursive Binary Search

```
public static int bsearch(int[] list, int target){
    return bsearch(list, target, 0, list.length - 1);
}

public static int bsearch(int[] list, int target,
                                  int first, int last){
    if( first <= last ){
        int mid = low + ((high - low) / 2);
        if( list[mid] == target )
            return mid;
        else if( list[mid] > target )
            return bsearch(list, target, first, mid - 1);
        else
            return bsearch(list, target, mid + 1, last);
    }
    return -1;
}
```

Other Searching Algorithms

- Interpolation Search
  - more like what people really do
- Indexed Searching
- Binary Search Trees
- Hash Table Searching
- Grover's Algorithm (Waiting for quantum computers to be built)
- best-first
- A*



qubits can be in a superposition of all the classically allowed states

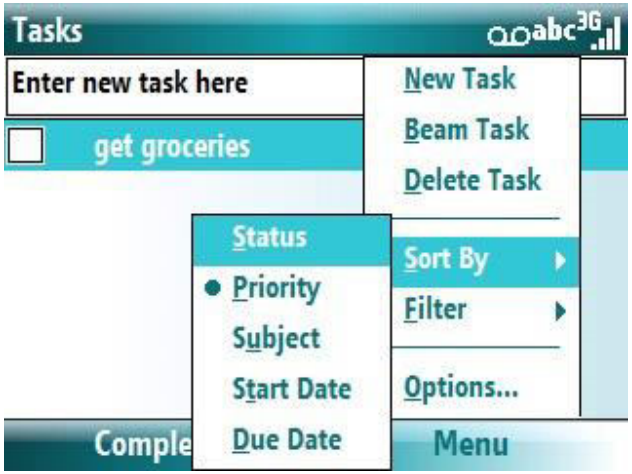
Sorting

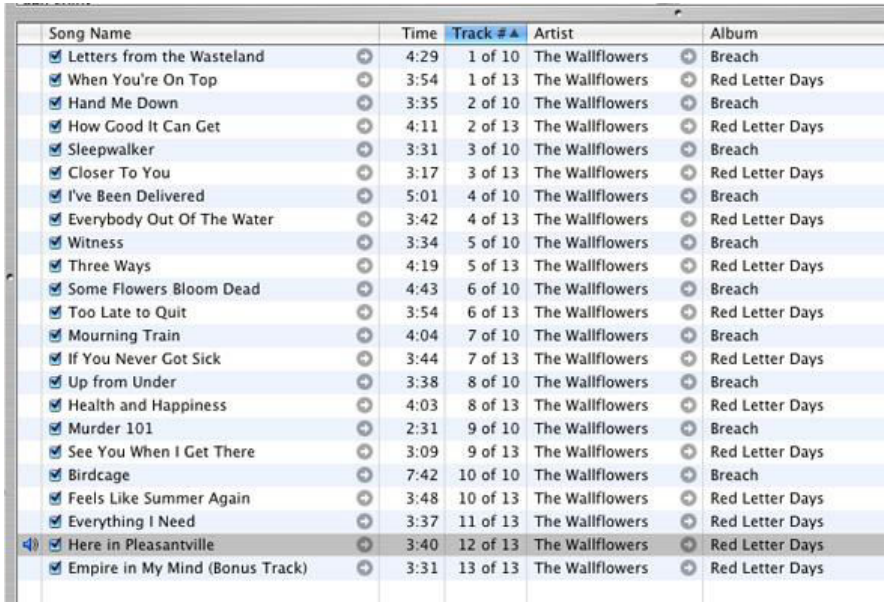
U.S. All-time List - Marathon

As of 4/24/08

Women

1	1	2:19:36	Deena Kastor nee Drossin
2		2:21:16	Drossin (2)
3	2	2:21:21	Joan Benoit Samuelson
4		2:21:25	Kastor (3)
5		2:22:43a	Benoit (2)
6		2:24:52a	Benoit (3)
7		2:26:11	Benoit (4)
8	3	2:26:26a	Julie Brown
9	4	2:26:40a	Kim Jones





Song Name	Time	Track #	Artist	Album
Letters from the Wasteland	4:29	1 of 10	The Wallflowers	Breach
When You're On Top	3:54	1 of 13	The Wallflowers	Red Letter Days
Hand Me Down	3:35	2 of 10	The Wallflowers	Breach
How Good It Can Get	4:11	2 of 13	The Wallflowers	Red Letter Days
Sleepwalker	3:31	3 of 10	The Wallflowers	Breach
Closer To You	3:17	3 of 13	The Wallflowers	Red Letter Days
I've Been Delivered	5:01	4 of 10	The Wallflowers	Breach
Everybody Out Of The Water	3:42	4 of 13	The Wallflowers	Red Letter Days
Witness	3:34	5 of 10	The Wallflowers	Breach
Three Ways	4:19	5 of 13	The Wallflowers	Red Letter Days
Some Flowers Bloom Dead	4:43	6 of 10	The Wallflowers	Breach
Too Late to Quit	3:54	6 of 13	The Wallflowers	Red Letter Days
Mourning Train	4:04	7 of 10	The Wallflowers	Breach
If You Never Got Sick	3:44	7 of 13	The Wallflowers	Red Letter Days
Up from Under	3:38	8 of 10	The Wallflowers	Breach
Health and Happiness	4:03	8 of 13	The Wallflowers	Red Letter Days
Murder 101	2:31	9 of 10	The Wallflowers	Breach
See You When I Get There	3:09	9 of 13	The Wallflowers	Red Letter Days
Birdcage	7:42	10 of 10	The Wallflowers	Breach
Feels Like Summer Again	3:48	10 of 13	The Wallflowers	Red Letter Days
Everything I Need	3:37	11 of 13	The Wallflowers	Red Letter Days
Here in Pleasantville	3:40	12 of 13	The Wallflowers	Red Letter Days
Empire in My Mind (Bonus Track)	3:31	13 of 13	The Wallflowers	Red Letter Days

Sorting Fun
Why Not Bubble Sort?

Sorting

- A fundamental application for computers
- Done to make finding data (searching) faster
- Many different algorithms for sorting
- One of the difficulties with sorting is working with a fixed size storage container (array)
  - if resize, that is expensive (slow)
- The "simple" sorts run in quadratic time $O(N^2)$
  - bubble sort
  - selection sort
  - insertion sort

Stable Sorting

- A property of sorts
- If a sort guarantees the relative order of equal items stays the same then it is a *stable sort*
- [7_1, 6, 7_2, 5, 1, 2, 7_3, -5]
  - subscripts added for clarity
- [-5, 1, 2, 5, 6, 7_1, 7_2, 7_3]
  - result of stable sort
- Real world example:
  - sort a table in Wikipedia by one criteria, then another
  - sort by country, then by major wins

Selection sort

- Algorithm
  - Search through the list and find the smallest element
  - swap the smallest element with the first element
  - repeat starting at second element and find the second smallest element

```
public static void selectionSort(int[] list)
{      int min;
       int temp;
       for(int i = 0; i < list.length - 1; i++) {
              min = i;
              for(int j = i + 1; j < list.length; j++)
                     if( list[j] < list[min] )
                            min = j;
              temp = list[i];
              list[i] = list[min];
              list[min] = temp;
        }
}
```

Selection Sort in Practice

44 68 191 119 119 37 83 82 191 45 158 130 76 153 39 25

What is the T(N), *actual* number of statements executed, of the selection sort code, given a list of N elements? What is the Big O?

Generic Selection Sort

```
public void selectionSort(Comparable[] list)
{   int min; Comparable temp;
    for(int i = 0; i < list.length - 1; i++) {
    {   min = i;
        for(int j = i + 1; j < list.length; j++)

            if( list[min].compareTo(list[j]) > 0 )
                min = j;
        temp = list[i];
        list[i] = list[min];
        list[min] = temp;
    }
}
```

- Best case, worst case, average case Big O?

Attendance Question 3

Is selection sort always stable?

A. Yes

B. No

Insertion Sort

- Another of the O(N^2) sorts
- The first item is sorted
- Compare the second item to the first
  - if smaller swap
- Third item, compare to item next to it
  - need to swap
  - after swap compare again
- And so forth…

Insertion Sort Code

```
public void insertionSort(int[] list)
{     int temp, j;
      for(int i = 1; i < list.length; i++)
      {     temp = list[i];
            j = i;
            while( j > 0 && temp < list[j - 1])
            {     // swap elements
                  list[j] = list[j - 1];
                  list[j - 1] = temp;
                  j--;
            }
      }
}
```

- Best case, worst case, average case Big O?

Attendance Question 4

- Is the version of insertion sort shown always stable?

A. Yes

B. No

Comparing Algorithms

- Which algorithm do you think will be faster given random data, selection sort or insertion sort?
- Why?

Sub Quadratic Sorting Algorithms

Sub Quadratic means having a Big O better than $O(N^2)$

ShellSort

- Created by Donald Shell in 1959
- Wanted to stop moving data small distances (in the case of insertion sort and bubble sort) and stop making swaps that are not helpful (in the case of selection sort)
- Start with sub arrays created by looking at data that is far apart and then reduce the gap size

ShellSort in practice

46 2 83 41 102 5 17 31 64 49 18

Gap of five. Sort sub array with 46, 5, and 18
5 2 83 41 102 18 17 31 64 49 46

Gap still five. Sort sub array with 2 and 17
5 2 83 41 102 18 17 31 64 49 46

Gap still five. Sort sub array with 83 and 31
5 2 31 41 102 18 17 83 64 49 46

Gap still five Sort sub array with 41 and 64
5 2 31 41 102 18 17 83 64 49 46

Gap still five. Sort sub array with 102 and 49
5 2 31 41 49 18 17 83 64 102 46

Continued on next slide:

Completed Shellsort

5 2 31 41 49 18 17 83 64 102 46

Gap now 2: Sort sub array with 5 31 49 17 64 46
5 2 17 41 31 18 46 83 49 102 64

Gap still 2: Sort sub array with 2 41 18 83 102
5 2 17 18 31 41 46 83 49 102 64

Gap of 1 (Insertion sort)

2 5 17 18 31 41 46 49 64 83 102

Array sorted

Shellsort on Another Data Set

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
44	68	191	119	119	37	83	82	191	45	158	130	76	153	39	25

Initial gap = length / 2 = 16 / 2 = 8

initial sub arrays indices:

{0, 8}, {1, 9}, {2, 10}, {3, 11}, {4, 12}, {5, 13}, {6, 14}, {7, 15}
next gap = 8 / 2 = 4

{0, 4, 8, 12}, {1, 5, 9, 13}, {2, 6, 10, 14}, {3, 7, 11, 15}
next gap = 4 / 2 = 2

{0, 2, 4, 6, 8, 10, 12, 14}, {1, 3, 5, 7, 9, 11, 13, 15}

final gap = 2 / 2 = 1

ShellSort Code

```
public static void shellsort(Comparable[] list)
{  Comparable temp; boolean swap;
   for(int gap = list.length / 2; gap > 0; gap /= 2)
      for(int i = gap; i < list.length; i++)
      {   Comparable tmp = list[i];
          int j = i;
          for( ; j >= gap &&
                 tmp.compareTo( list[j - gap] ) < 0;
                 j -= gap )
             list[ j ] = list[ j - gap ];
          list[ j ] = tmp;
       }
}
```

Comparison of Various Sorts

Num Items	Selection	Insertion	Shellsort	Quicksort
1000	16	5	0	0
2000	59	49	0	6
4000	271	175	6	5
8000	1056	686	11	0
16000	4203	2754	32	11
32000	16852	11039	37	45
64000	expected?	expected?	100	68
128000	expected?	expected?	257	158
256000	expected?	expected?	543	335
512000	expected?	expected?	1210	722
1024000	expected?	expected?	2522	1550

times in milliseconds

Quicksort

- Invented by C.A.R. (Tony) Hoare
- A divide and conquer approach that uses recursion

1. If the list has 0 or 1 elements it is sorted
2. otherwise, pick any element p in the list. This is called the pivot value
3. Partition the list minus the pivot into two sub lists according to values less than or greater than the pivot. (equal values go to either)
4. return the quicksort of the first list followed by the quicksort of the second list

Quicksort in Action

39 23 17 90 33 72 46 79 11 52 64 5 71

Pick middle element as pivot: 46

Partition list

23 17 5 33 39 11 46 79 72 52 64 90 71

quick sort the less than list

Pick middle element as pivot: 33

23 17 5 11 33 39

quicksort the less than list, pivot now 5

{} 5 23 17 11

quicksort the less than list, base case

quicksort the greater than list

Pick middle element as pivot: 17

and so on….

Quicksort on Another Data Set

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
44	68	191	119	119	37	83	82	191	45	158	130	76	153	39	25

Big O of Quicksort?

```
public static void swapReferences( Object[] a, int index1, int index2 )
{   Object tmp = a[index1];
    a[index1] = a[index2];
    a[index2] = tmp;
}

 public void quicksort( Comparable[] list, int start, int stop )
 {    if(start >= stop)
        return; //base case list of 0 or 1 elements

      int pivotIndex = (start + stop) / 2;

      // Place pivot at start position
      swapReferences(list, pivotIndex, start);
      Comparable pivot = list[start];

      // Begin partitioning
      int i, j = start;

      // from first to j are elements less than or equal to pivot
      // from j to i are elements greater than pivot
      // elements beyond i have not been checked yet
      for(i = start + 1; i <= stop; i++ )
      {   //is current element less than or equal to pivot
          if(list[i].compareTo(pivot) <= 0)
          {   // if so move it to the less than or equal portion
              j++;
              swapReferences(list, i, j);
          }
      }

      //restore pivot to correct spot
      swapReferences(list, start, j);
      quicksort( list, start, j - 1 );    // Sort small elements
      quicksort( list, j + 1, stop );   // Sort large elements
  }
```

Attendance Question 5

- What is the best case and worst case Big O of quicksort?

	Best	Worst
A.	O(NlogN)	$O(N^2)$
B.	$O(N^2)$	$O(N^2)$
C.	$O(N^2)$	O(N!)
D.	O(NlogN)	O(NlogN)
E.	O(N)	O(NlogN)

Quicksort Caveats

- Average case Big O?
- Worst case Big O?
- Coding the partition step is usually the hardest part

Attendance Question 6

- You have 1,000,000 items that you will be searching. How many searches need to be performed before the data is changed to make sorting worthwhile?

A. 10

B. 40

C. 1,000

D. 10,000

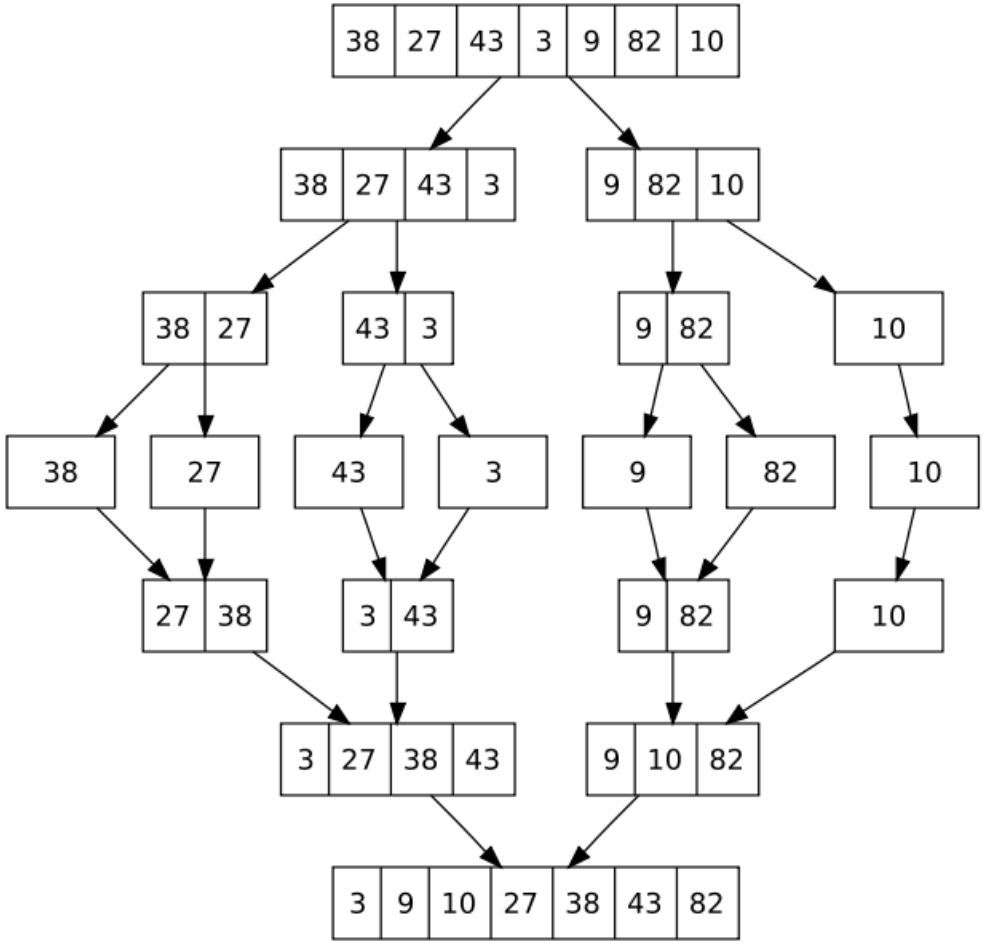
E. 500,000

Merge Sort Algorithm

Don Knuth cites John von Neumann as the creator of this algorithm

1. If a list has 1 element or 0 elements it is sorted
2. If a list has more than 2 split into into 2 separate lists
3. Perform this algorithm on each of those smaller lists
4. Take the 2 sorted lists and merge them together

Merge Sort



When implementing one temporary array is used instead of multiple temporary arrays.

Why?

Merge Sort code

```
/**
 * perform a merge sort on the data in c
 * @param c c != null, all elements of c
 * are the same data type
 */
public static void mergeSort(Comparable[] c)
{       Comparable[] temp = new Comparable[ c.length ];
        sort(c, temp, 0, c.length - 1);
}

private static void sort(Comparable[] list, Comparable[] temp,
                                int low, int high)
{       if( low < high){
                int center = (low + high) / 2;
                sort(list, temp, low, center);
                sort(list, temp, center + 1, high);
                merge(list, temp, low, center + 1, high);
        }
}
```

Merge Sort Code

```
private static void merge( Comparable[] list, Comparable[] temp,
                           int leftPos, int rightPos, int rightEnd){
        int leftEnd = rightPos - 1;
        int tempPos = leftPos;
        int numElements = rightEnd - leftPos + 1;
        //main loop
        while( leftPos <= leftEnd && rightPos <= rightEnd){
                if( list[ leftPos ].compareTo(list[rightPos]) <= 0){
                        temp[ tempPos ] = list[ leftPos ];
                        leftPos++;
                }
                else{
                        temp[ tempPos ] = list[ rightPos ];
                        rightPos++;
                }
                tempPos++;
        }
        //copy rest of left half
        while( leftPos <= leftEnd){
                temp[ tempPos ] = list[ leftPos ];
                tempPos++;
                leftPos++;
        }
        //copy rest of right half
        while( rightPos <= rightEnd){
                temp[ tempPos ] = list[ rightPos ];
                tempPos++;
                rightPos++;
        }
        //Copy temp back into list
        for(int i = 0; i < numElements; i++, rightEnd--)
                list[ rightEnd ] = temp[ rightEnd ];
}
```

Final Comments

- Language libraries often have sorting algorithms in them
  - Java Arrays and Collections classes
  - C++ Standard Template Library
  - Python sort and sorted functions
- Hybrid sorts
  - when size of unsorted list or portion of array is small use insertion sort, otherwise use O(N log N) sort like Quicksort of Mergesort
- Many other sorting algorithms exist.