Doing Science

To write like a scientist, you first have to think like one

Applying scientific thinking to improving mergesort

Thinking like a Scientist

A rancher hired an engineer, a scientist and a mathematician to build a fence ....

The 9s of uptime

Number of 9s	% uptime	downtime in a year
1	90%	36 days
2	99%	3.6 days 86 hours
3	99.9	8.6 hours 500 minutes
4	99.99	50 minutes
5	99.999	5 minutes 300 seconds
6	99.9999	30 seconds
7	99.99999	3 seconds

Phone system with 6 Nines of uptime

- 2 computers: 1 live and one spare
  - must be several miles apart
  - idea: replicate memory from live to spare
    - if time between replications is N seconds, then need to be able to:
    - A. identify all RAM that has changed in the past second
    - B. transmit those changes
    - C. update the "spare"
      - time for A+B < N seconds
      - time for C less than N seconds
    - If A+B+C < 3 seconds than can get
    - **7 Nines -- assuming only one transition per year**
    - **6 Nines -- assuming 10 transitions!**
      - **and we can do it on commodity hardware**

The Engineering Approach

- I was working with a bunch of engineers
  - they spec'd the problem,
    - determined max speed of transmission between two computers 5 miles apart
      - start at M seconds: Ask: does it work? Is is good enough?
      - repeat until either "does it work" is NO or "good enough" is YES
    - Conclusion: at 200 (ish) ms it still worked and was deemed "good enough"
- At that rate 7 Nines seems achievable!!!

The Science Approach

- I asked "What is the shortest replication interval achievable and why"
  - How do I ask this question???

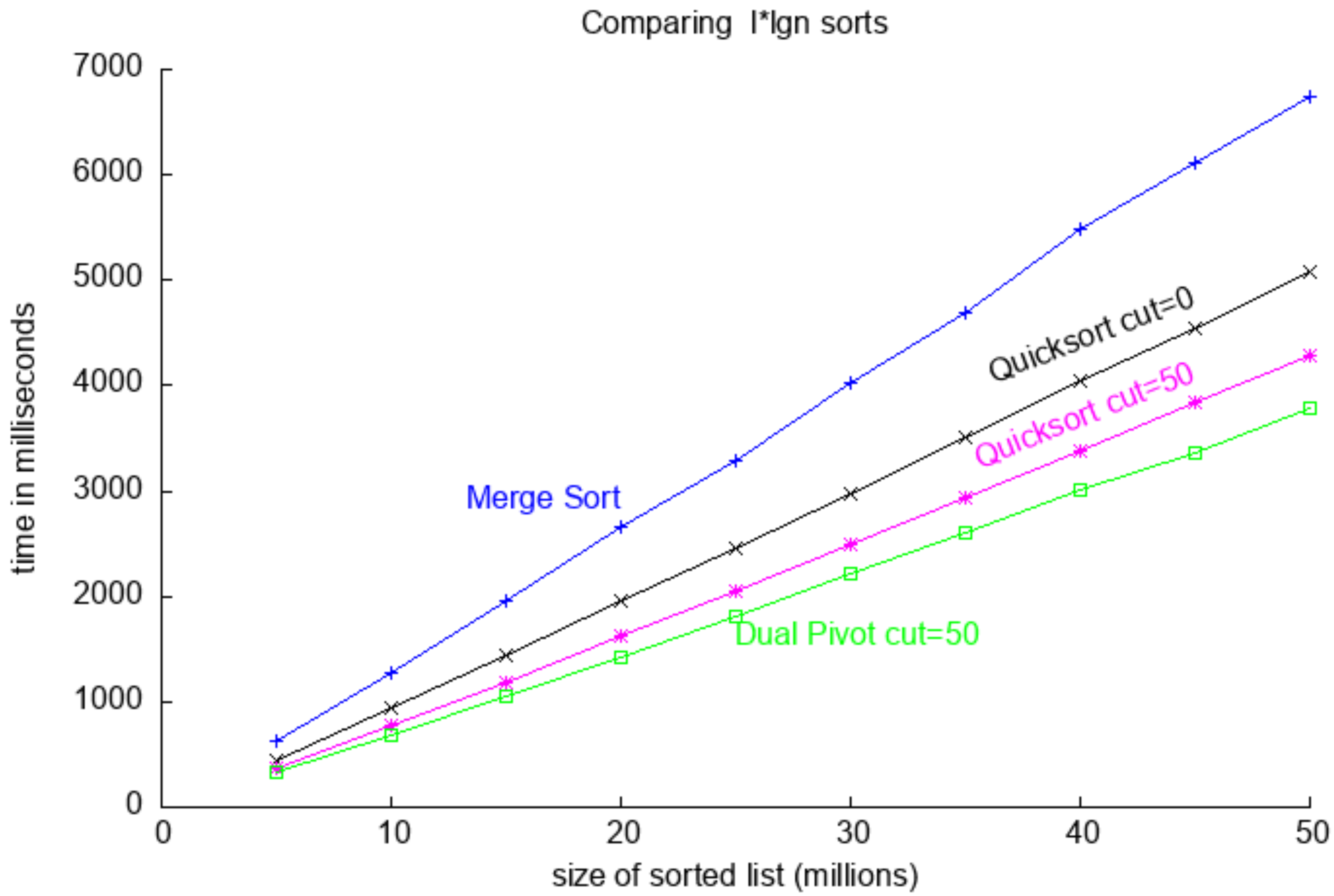
- What do I know?
- What data can I get?
  - ie what is knowable?

What is the shortest replication interval achievable and why

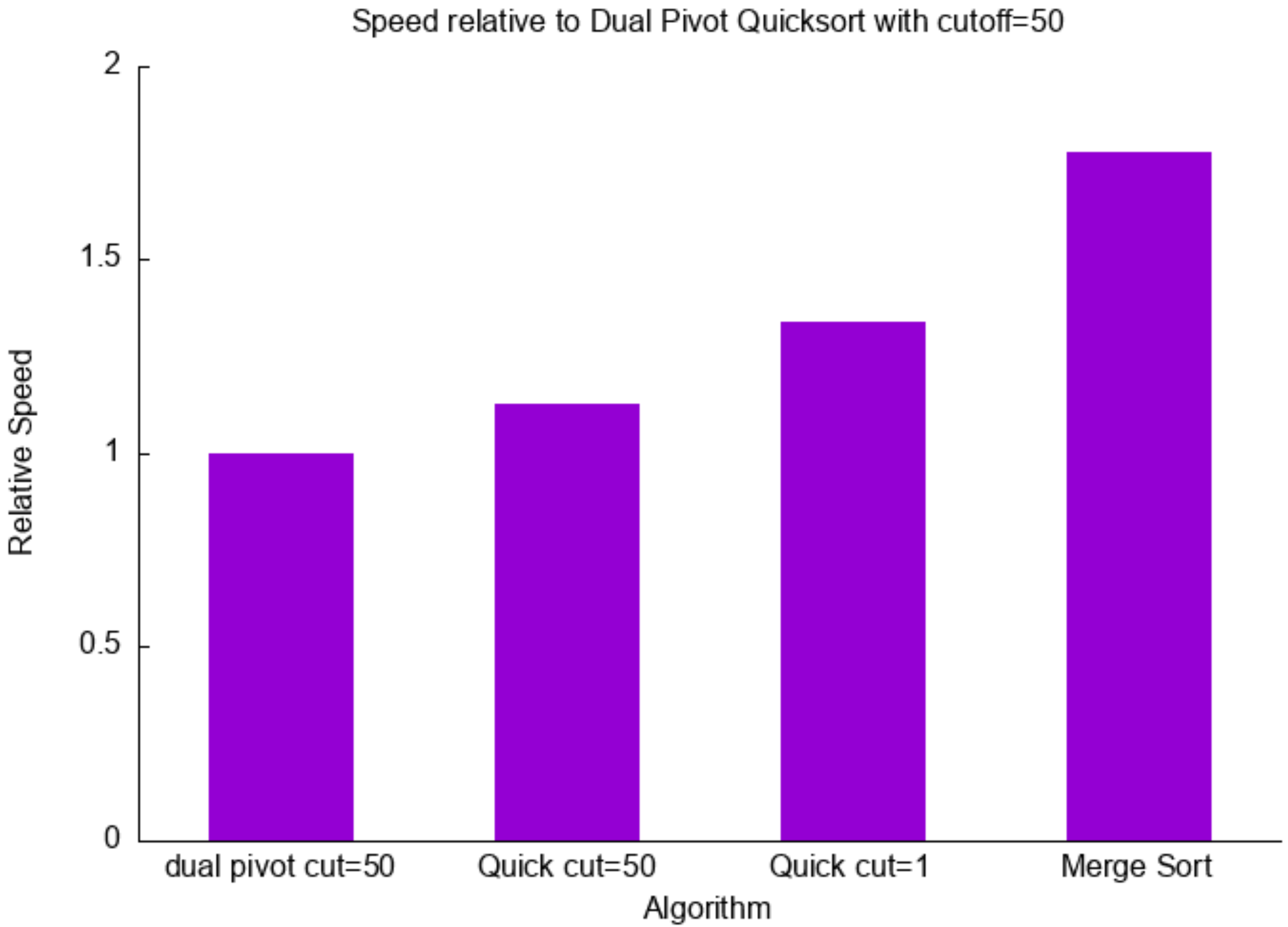
- Known: transmission rate: Mbits/second
- Can ask: given a time start how much memory has changed between
- So in 2 ms intervals over the corse of several days on a phone server
  - On average how much has changed:
    - in 2ms
    - in 4ms
    - in 8ms
    - ...

Think like a scientist about Mergesort

Hypothesis: I can speed up Mergesort



Comparing I*lgn sorts
7000
6000
5000
4000
3000
2000
1000
0
0
10
20
30
40
50
time in milliseconds
size of sorted list (millions)
Merge Sort
Quicksort cut=0
Quicksort cut=50
Dual Pivot cut=50

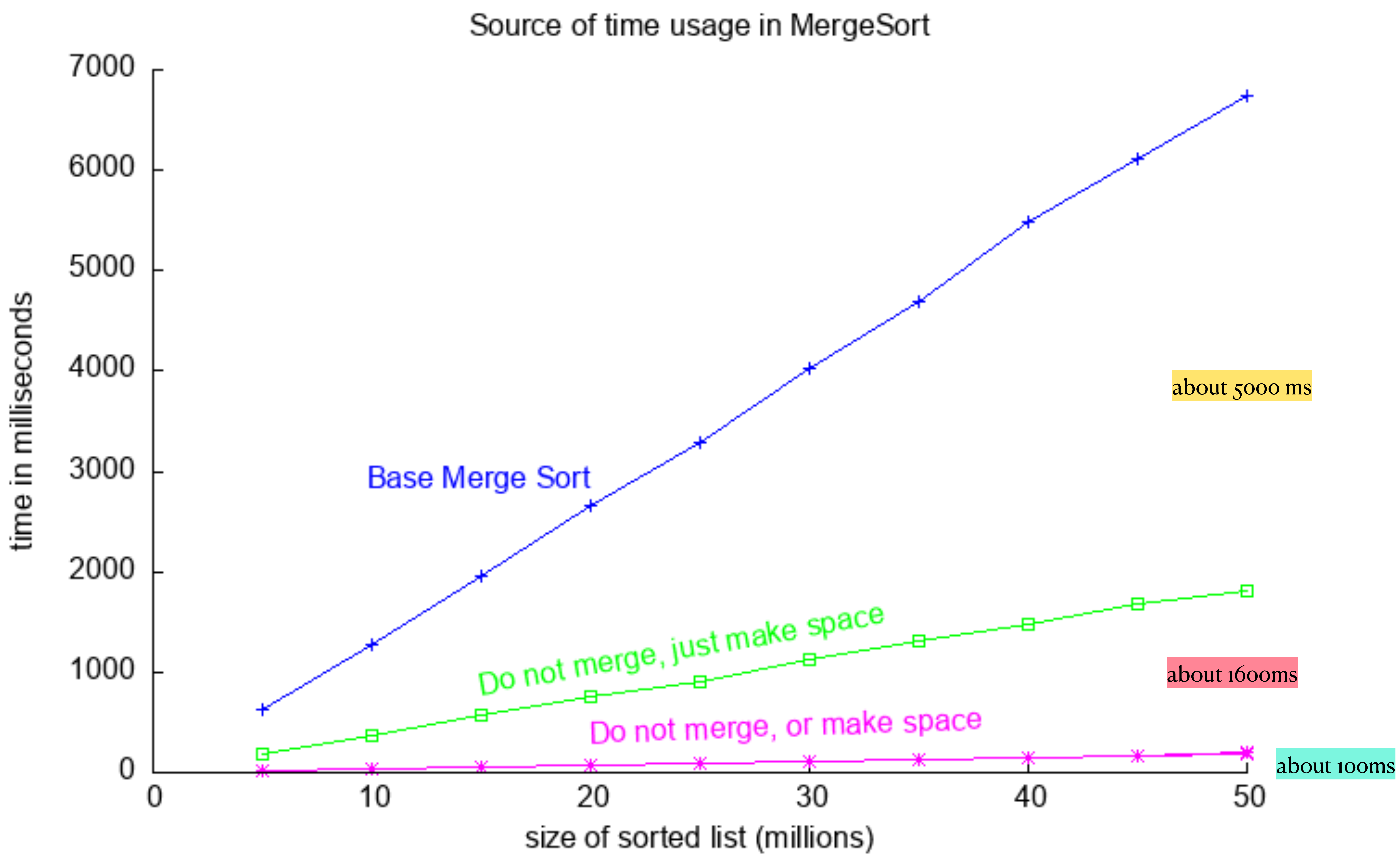

Speed relative to Dual Pivot Quicksort with cutoff=50
2
1.5
1
0.5
0
Relative Speed
dual pivot cut=50
Quick cut=50
Quick cut=1
Merge Sort
Algorithm

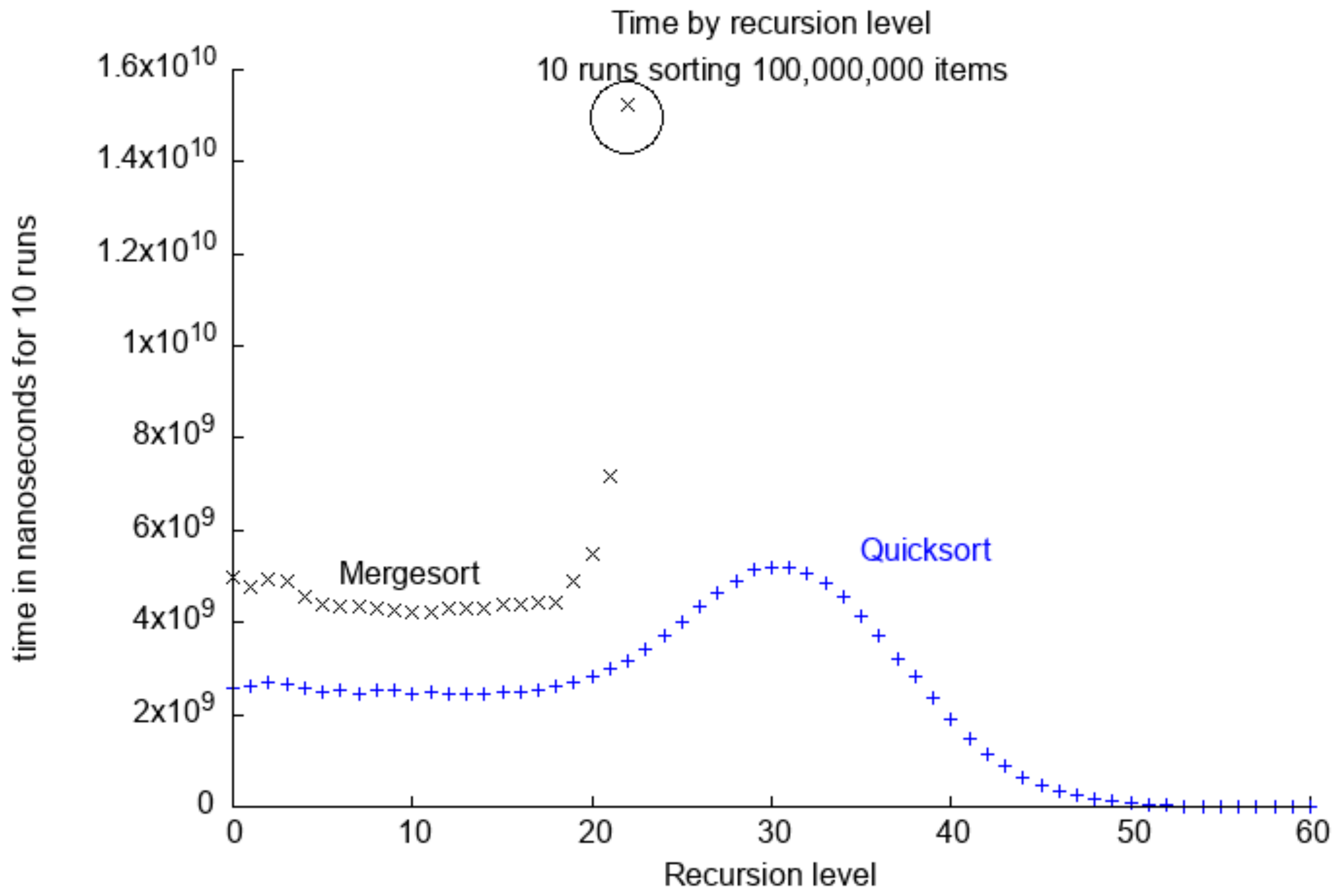
Question: Where is the time used in MergeSort?

Assume getting half takes 0 time

```
MergeSort(list)
    if length(list)<=1 return list
    return merge(MergeSort(half1),
                 MergeSort(half2))

merge(l1, l2)
    let nArray = new [l1.len+l2.len]
    merge l1 and l2 in nArray
    return nArray
```

3 pieces of the algorithm require time


Source of time usage in MergeSort
7000
6000
5000
4000
3000
2000
1000
0
time in milliseconds
0
10
20
30
40
50
size of sorted list (millions)
Base Merge Sort
Do not merge, just make space
Do not merge, or make space
about 5000 ms
about 1600ms
about 100ms

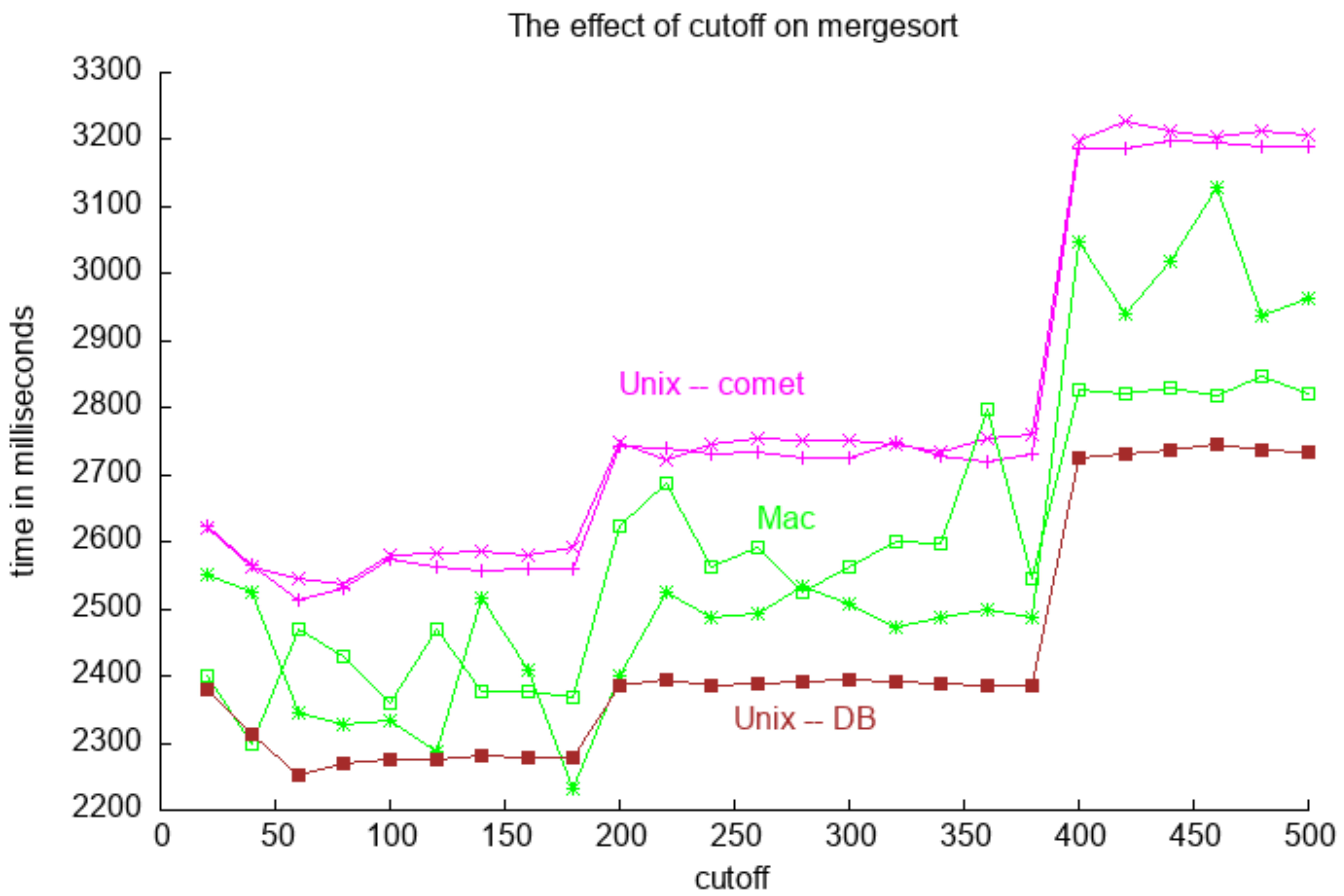

Time by recursion level
10 runs sorting 100,000,000 items
time in nanoseconds for 10 runs
Recursion level
Mergesort
Quicksort
0
2x10^9
4x10^9
6x10^9
8x10^9
1x10^10
1.2x10^10
1.4x10^10
1.6x10^10
0
10
20
30
40
50
60

Merge Sort With Insertion Sort on small chunks

MergeSort is not "in place"
So you need to return the sorted array.

```
func domerge(list1, list2 []int) []int {
    rtn:=make([]int, len(list1)+len(list2))
    merge into rtn
    return rtn;
}

func doMergeSort(list []int) []int {
    if len(list) <= 1 {
        return list
    }
    if len(list) < cutoff {
        iSort(list, 0, len(list)-1)
        return list
    }
    mid := len(list)/2
    return domerge(doMergeSort(list[:mid]), doMergeSort(list[mid:]));
}
```

Here is the insertion sort.
This is the only change
from standard mergeSort


The effect of cutoff on mergesort
time in milliseconds
cutoff
Unix -- comet
Mac
Unix -- DB
3300
3200
3100
3000
2900
2800
2700
2600
2500
2400
2300
2200
0
50
100
150
200
250
300
350
400
450
500

Merge Sort With a backup array

Merge merges from source array into target array

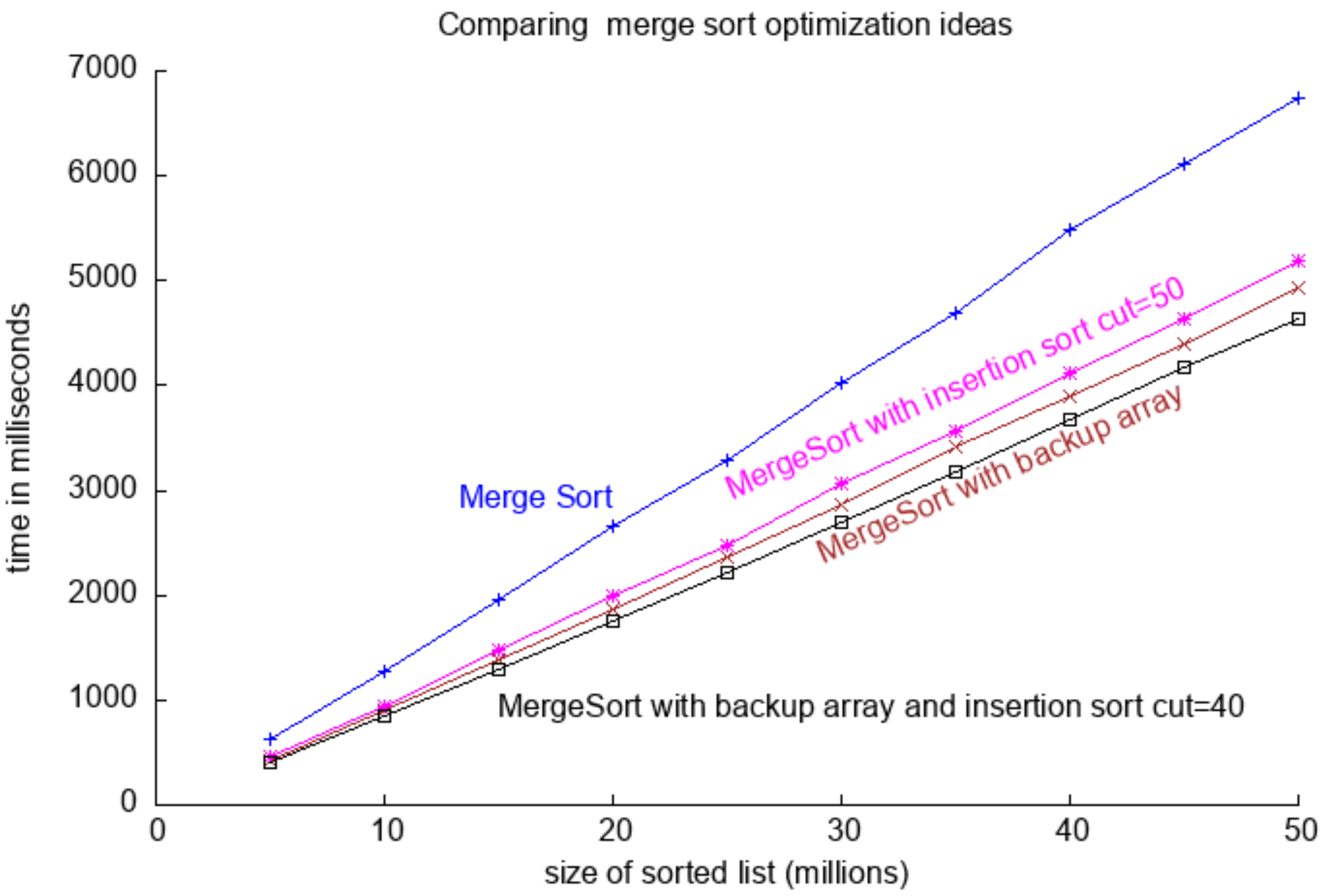
```
func mmerge(source, target []int, start, gap int) {
   // merge happens here
   }

func mimergeSort(list []int, left, right int)  []int {
    if cutoff>1 {
        for a:=0; a<len(list); a+=cutoff {
            b:=a+cutoff-1
            if b>=len(list) {
                b=len(list)-1
            }
            iSort(list, a, b)
        }
    }
    z:=cutoff
    if z<1 { z=1 }
    A := list
    B := make([]int, len(list))
    for ;z<len(list); z=z*2 {
        for aa:=0; aa<len(A); aa+=z*2 {
            mmerge(A, B, aa, z)
        }
        A,B = B,A  // swap
   }
   return A
}
```

Because we are going to use a backup array, we can do merge sort without all of that annoying recursion and splitting.

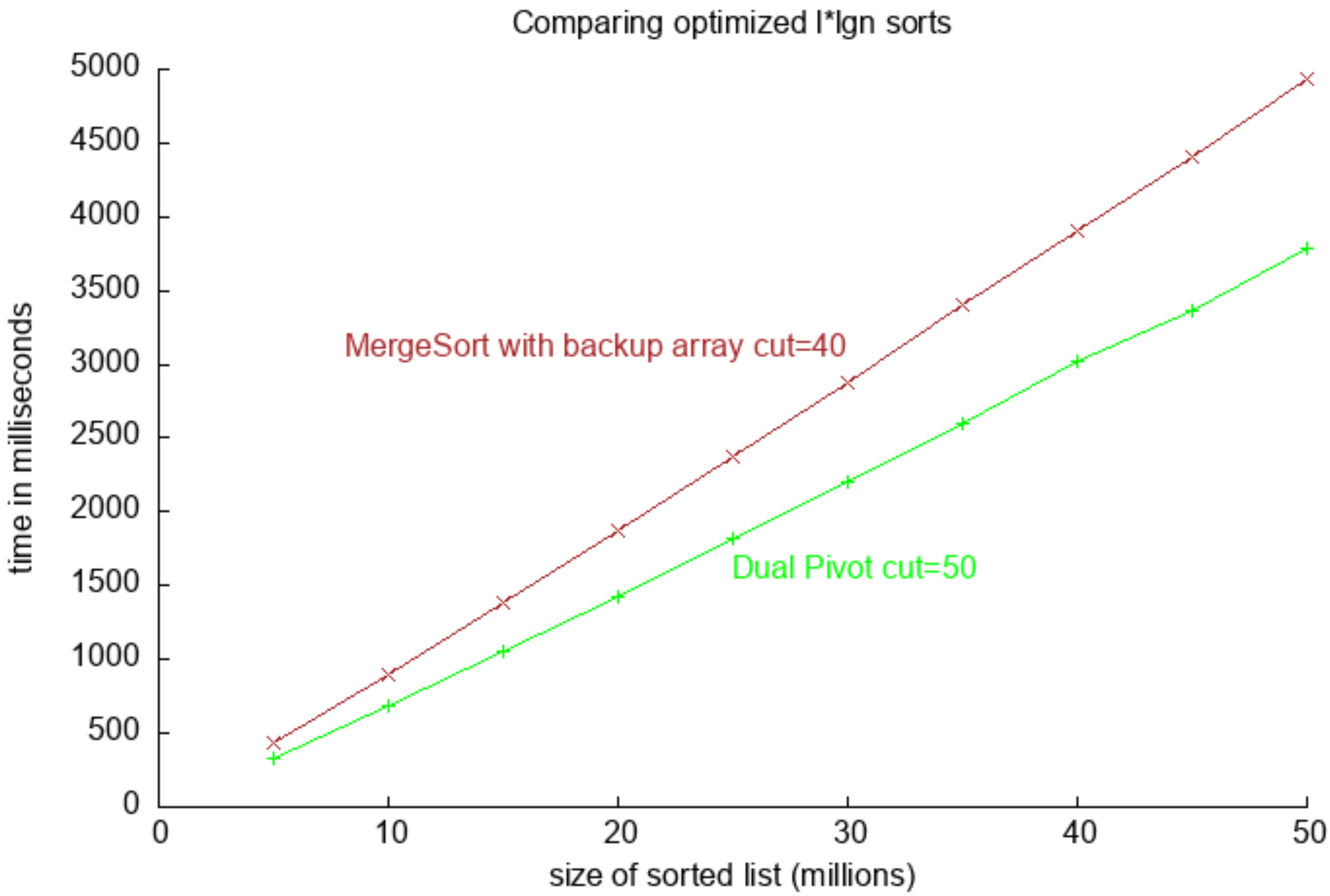
Here is the insertion sort. Since we have a single array, we use insertion sort of each group of size "cutoff" so each little group is sorted

Make the backup array. Do that once

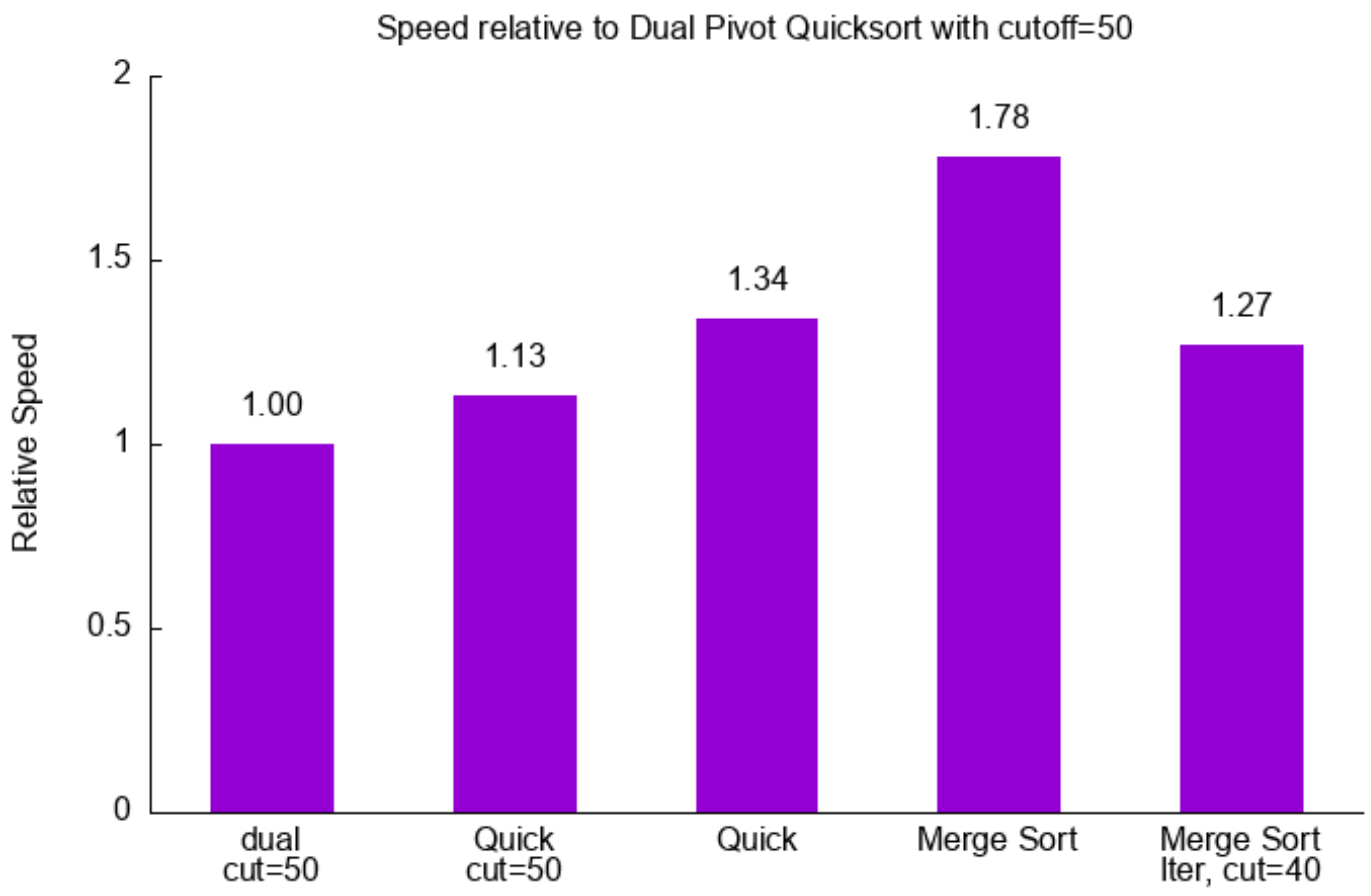
No splitting, just merging



Comparing merge sort optimization ideas
time in milliseconds
size of sorted list (millions)
0
1000
2000
3000
4000
5000
6000
7000
0
10
20
30
40
50
Merge Sort
MergeSort with insertion sort cut=50
MergeSort with backup array
MergeSort with backup array and insertion sort cut=40



Comparing optimized l*lgn sorts
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
time in milliseconds
0
10
20
30
40
50
size of sorted list (millions)
MergeSort with backup array cut=40
Dual Pivot cut=50


Speed relative to Dual Pivot Quicksort with cutoff=50
Relative Speed
2
1.5
1
0.5
0
1.00
1.13
1.34
1.78
1.27
dual cut=50
Quick cut=50
Quick
Merge Sort
Merge Sort Iter, cut=40

When sorting random integers
(in my Go implementation)

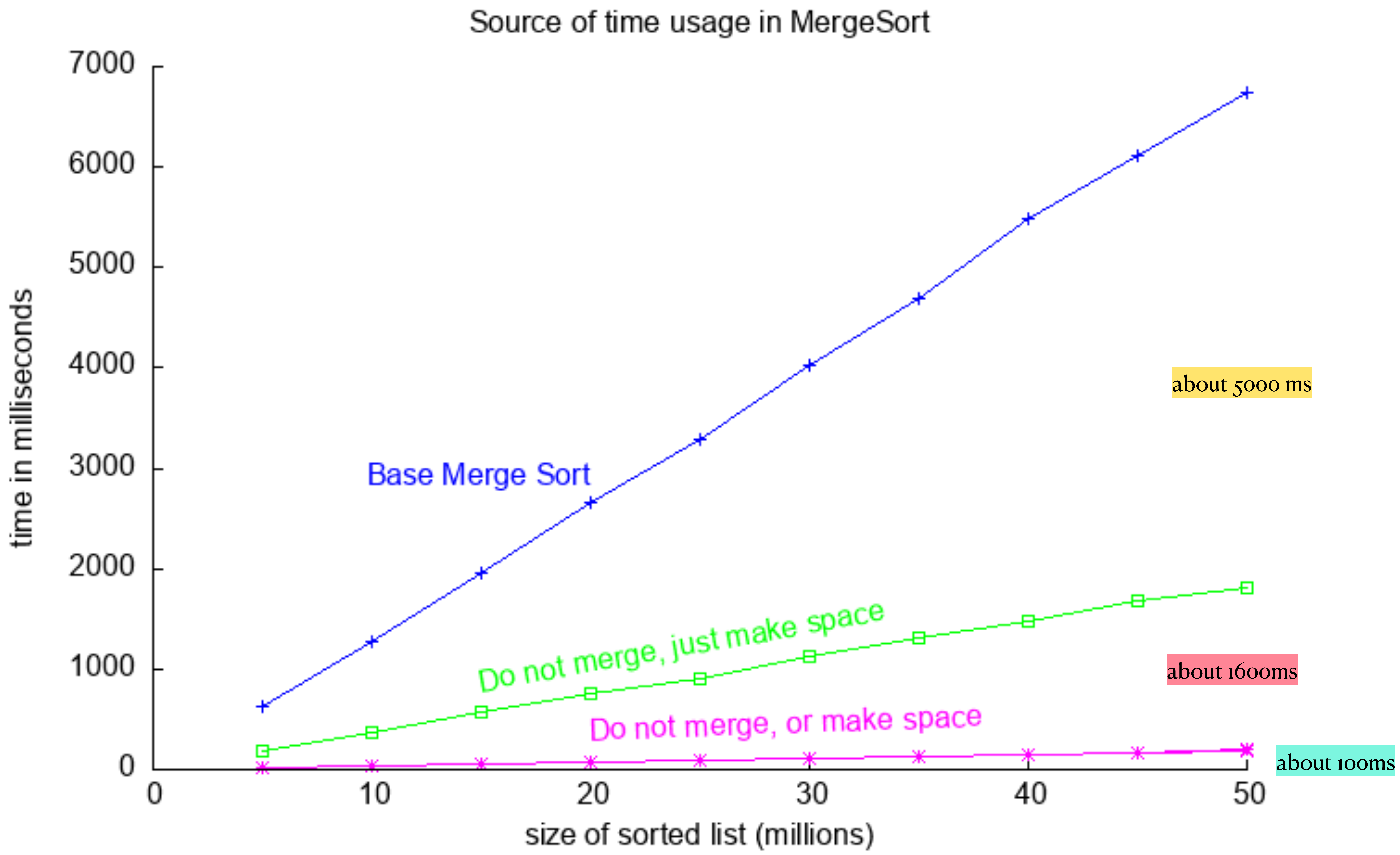
Rank by speed:

Dual pivot (cut=50)

Quicksort (cut=50)

Mergesort (cut=40)

Why is mergesort slower?


Source of time usage in MergeSort
7000
6000
5000
4000
3000
2000
1000
0
time in milliseconds
Base Merge Sort
Do not merge, just make space
Do not merge, or make space
about 5000 ms
about 1600ms
about 100ms
0
10
20
30
40
50
size of sorted list (millions)

Partition is faster than merge

MergeSort merge	Quicksort partition

```go
func mmerge(source, target []int, start, gap int) {
    locr:=start
    loc1:=start
    end1:=start+gap

    loc2:=start+gap
    end2:=start+2*gap
    if end2>=len(target) {
        end2=len(target)
    }
    for ;loc1<end1 && loc2<end2; {
        if source[loc1]>source[loc2] {
            target[locr]=source[loc2];
            loc2++
        } else {
            target[locr]=source[loc1];
            loc1++
        }
        locr++
    }
    for i:=loc1; i<end1; i++ {
        target[locr]=source[i];
        locr++
    }
    for i:=loc2; i<end2; i++ {
        target[locr]=source[i];
        locr++;
    }
}
```

```go
m:=left
for i:=left+1; i<=right; i++ {
    if (A[i] < A[left]) {
        m++
        A[i],A[m] = A[m],A[i]  //swap
    }
}
```