CMPUT 379 ASSIGNMENT 3 REPORT

By:JiaYi Che
CCID: jche

Figures Page

To avoid images take too much space I will put all the image here and give them figure sign to identify each image.

(note: Figure 1 to 24 all have 11 as the test program input)

Figure 1

Heapsort input 11, no skip, page size 4096, windows size 100

Mean: 7.6, variance: 1.5

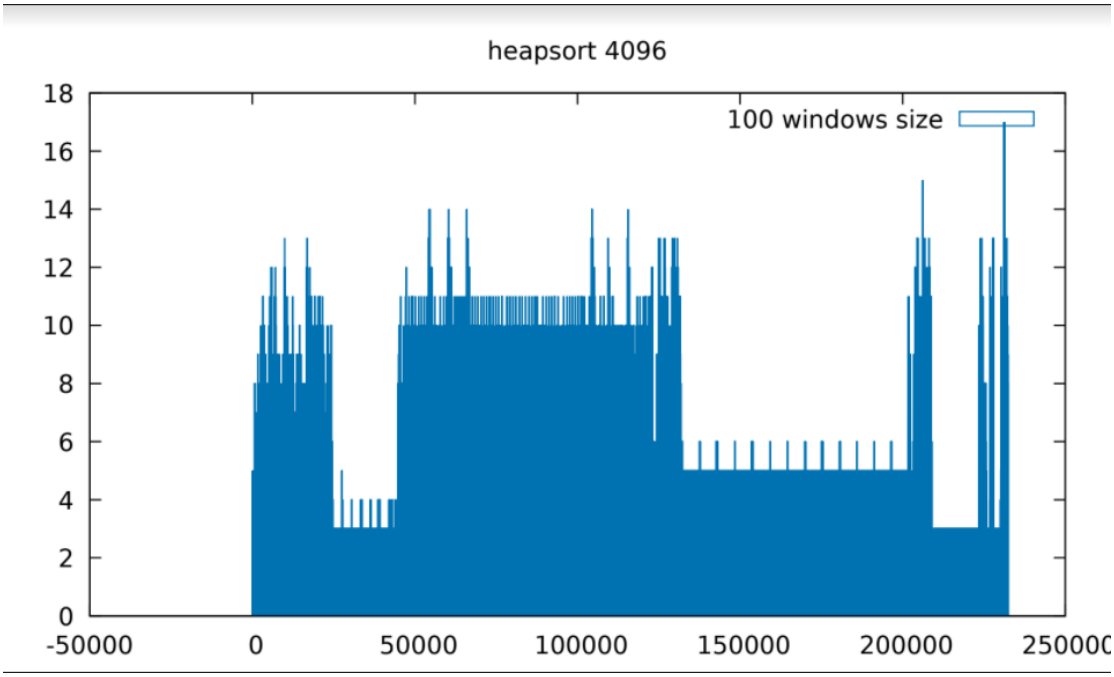


Figure 2
Heapsort input 11, no skip, page size 4096, windows size 1000
Mean: 13.4, variance: 3.6

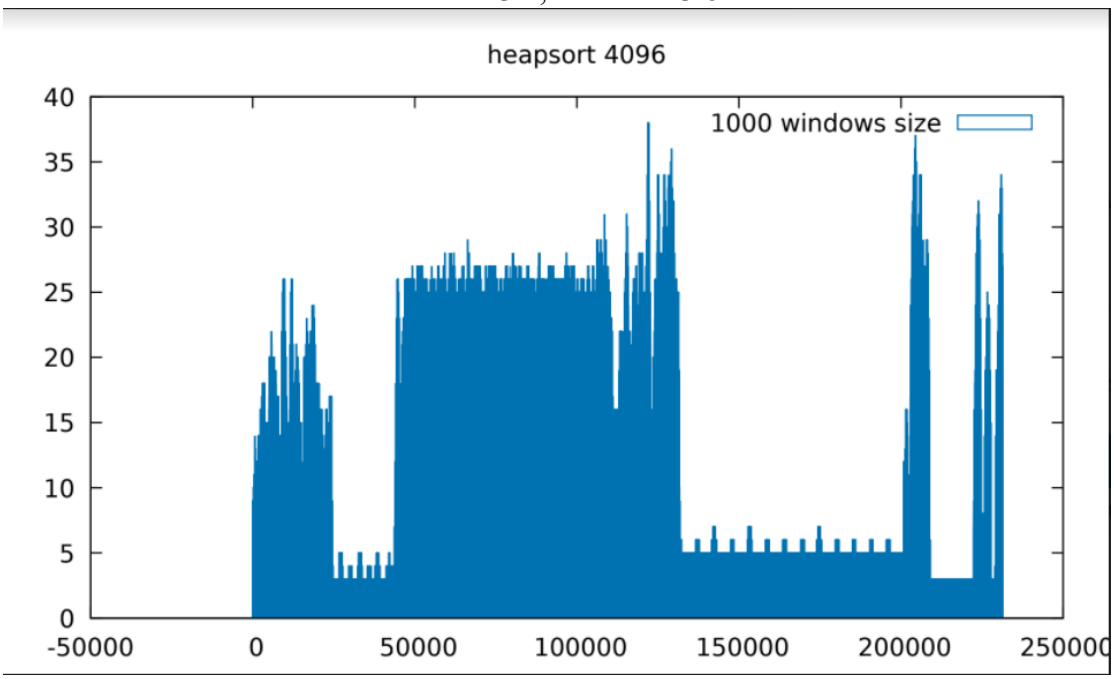


Figure 3
Heapsort input 11, no skip, page size 4096, windows size 10000
Mean: 37.5, variance: 6.3

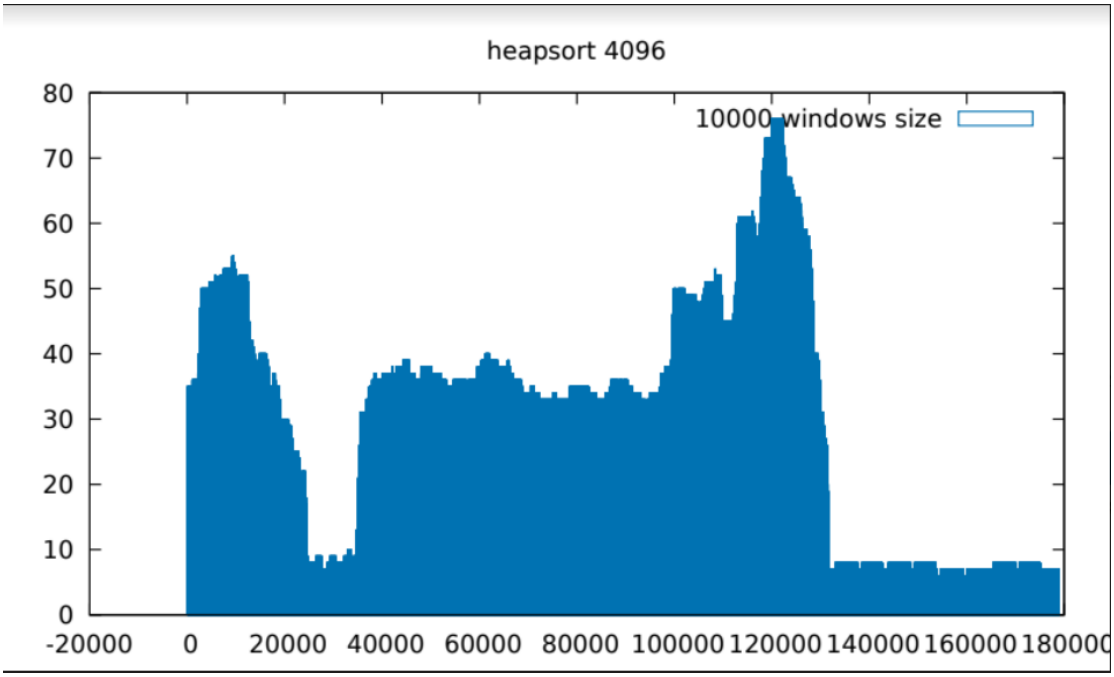


Figure 4
Heapsort input 11, no skip, page size 8192, windows size 100
Mean: 6, variance: 1.3

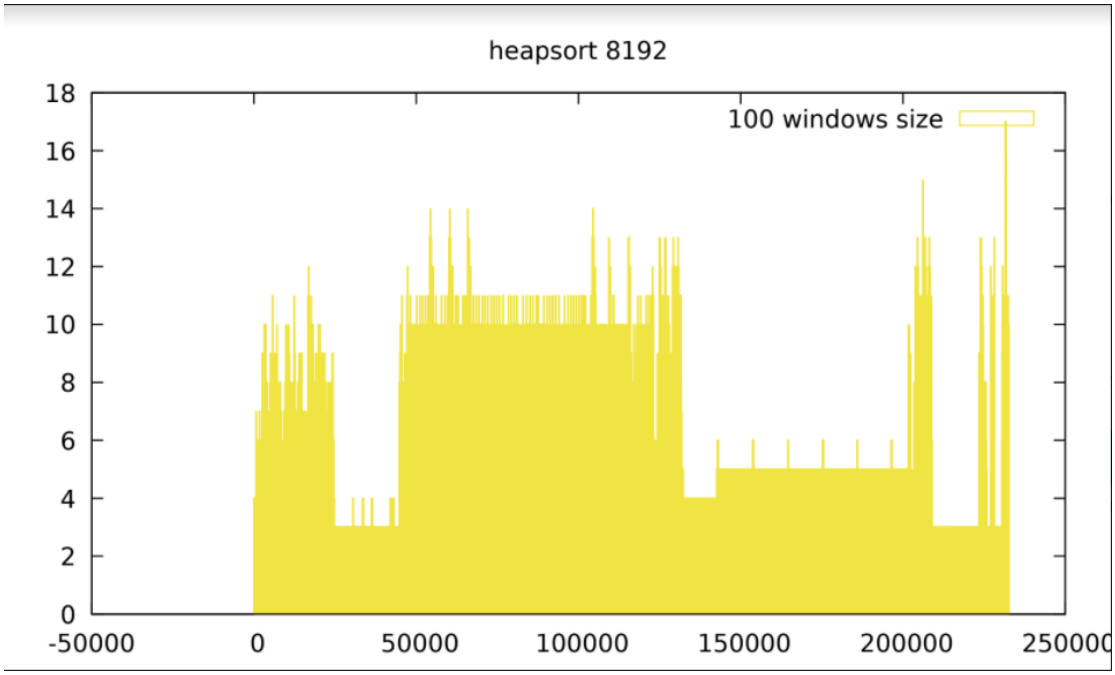


Figure 5
Heapsort input 11, no skip, page size 8192, windows size 1000
Mean: 12.8, variance: 2.6

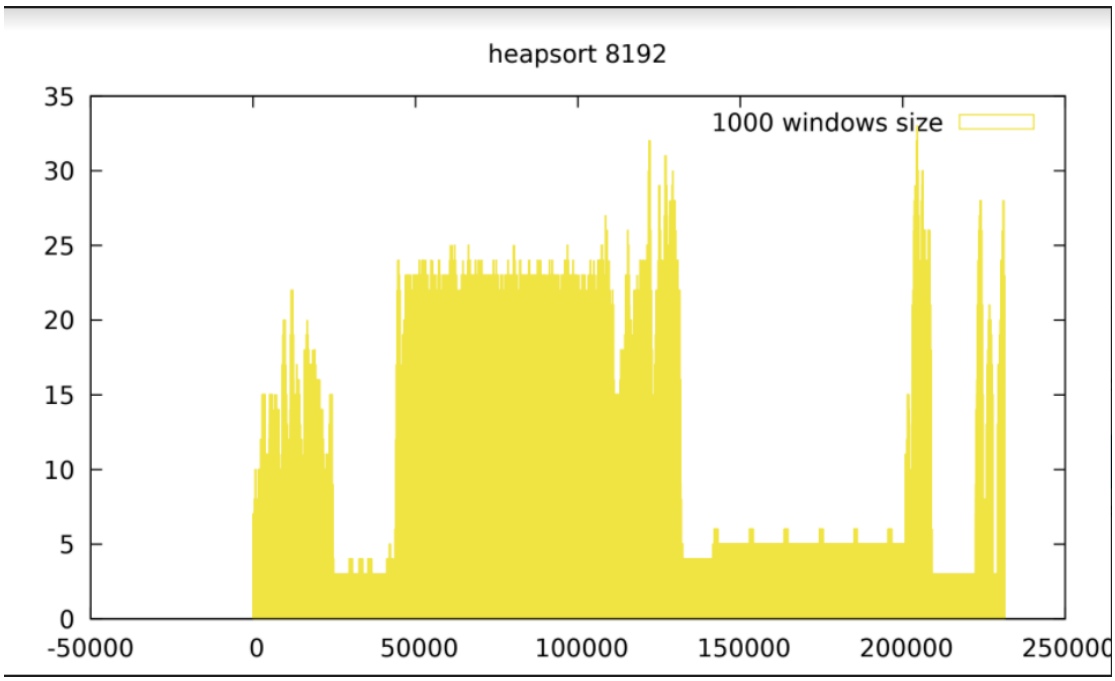


Figure 6
Heapsort input 11, no skip, page size 8192, windows size 10000
Mean: 24.7, variance: 5.6

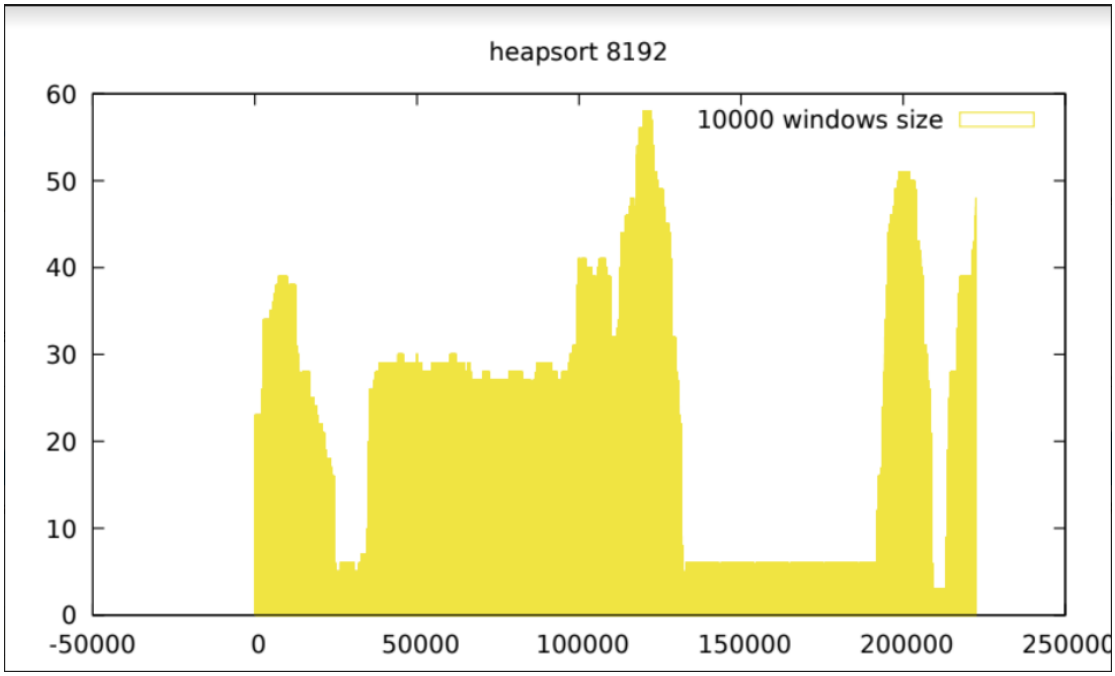


Figure 7
Quicksort input 11, no skip, page size 4096, windows size 100
Mean: 6, variance: 2.3

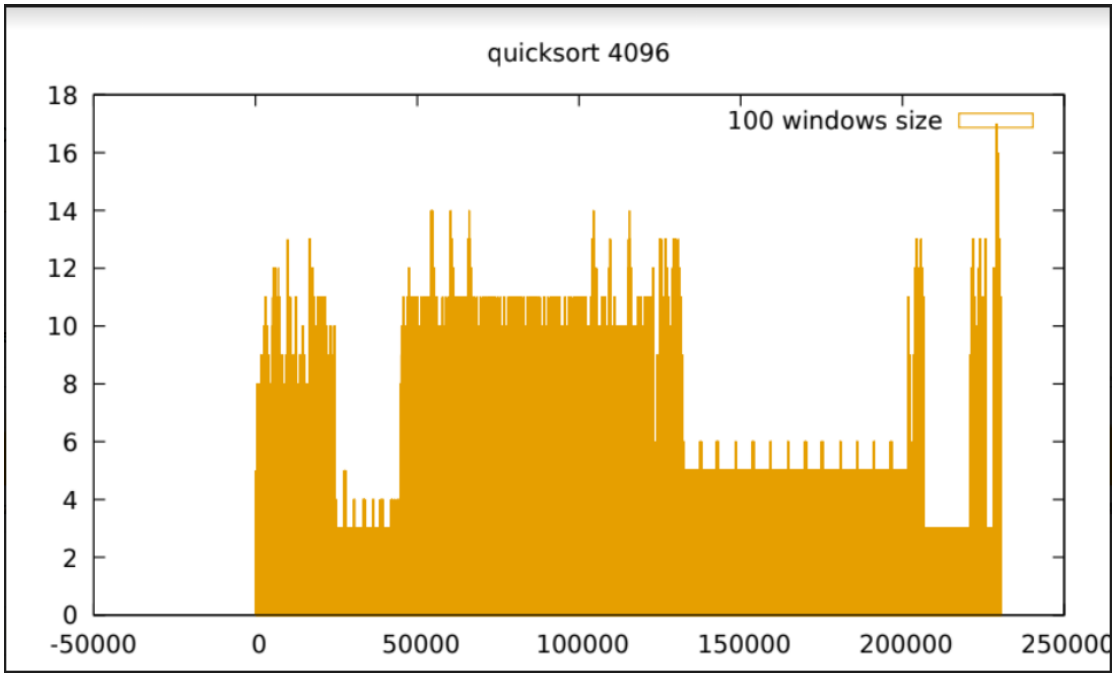


Figure 8
Quicksort input 11, no skip, page size 4096, windows size 1000
Mean: 15, variance: 4.5

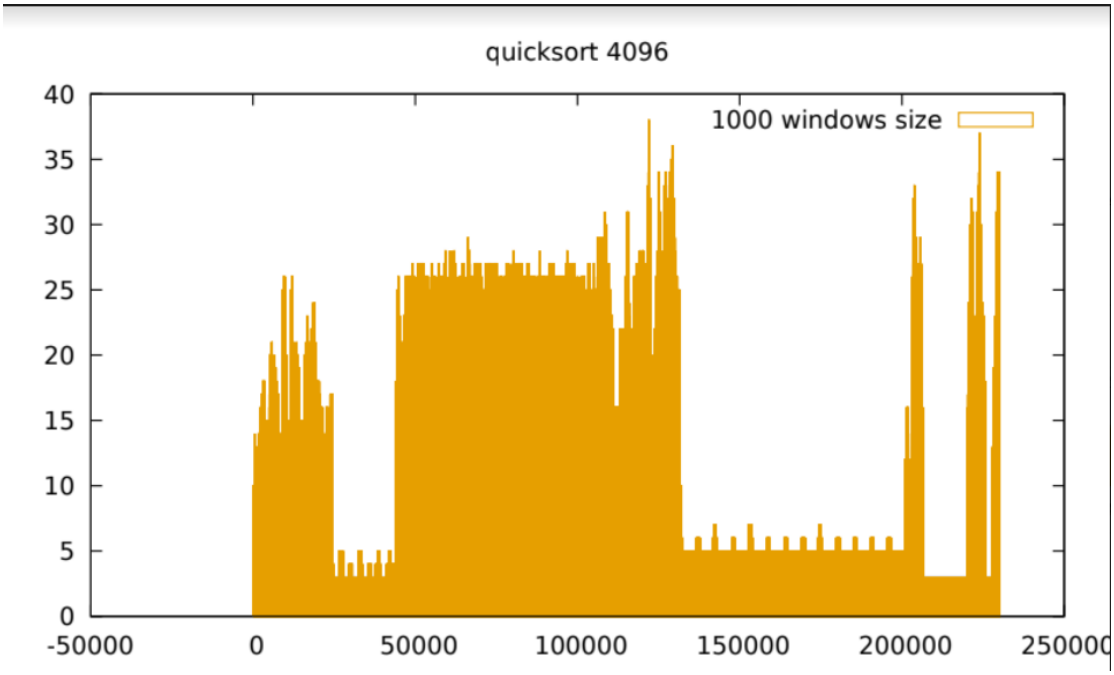


Figure 9
Quicksort input 11, no skip, page size 4096, windows size 10000
Mean: 30, variance: 6.2

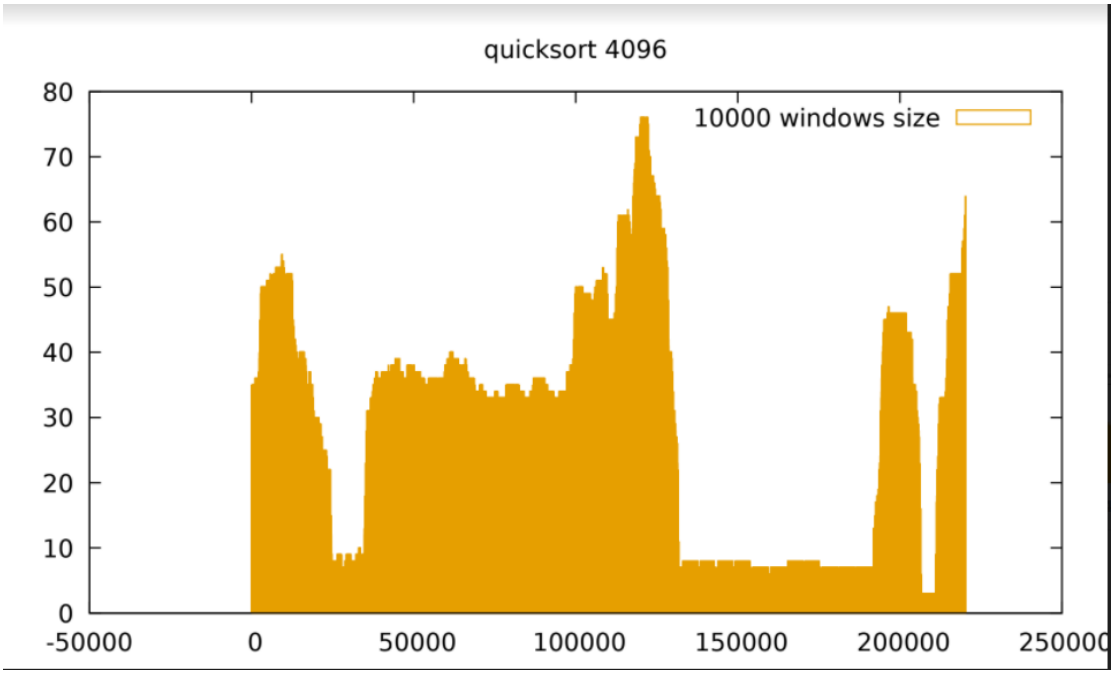


Figure 10
Quicksort input 11, no skip, page size 8192, windows size 100
Mean: 6.4, variance: 1.6

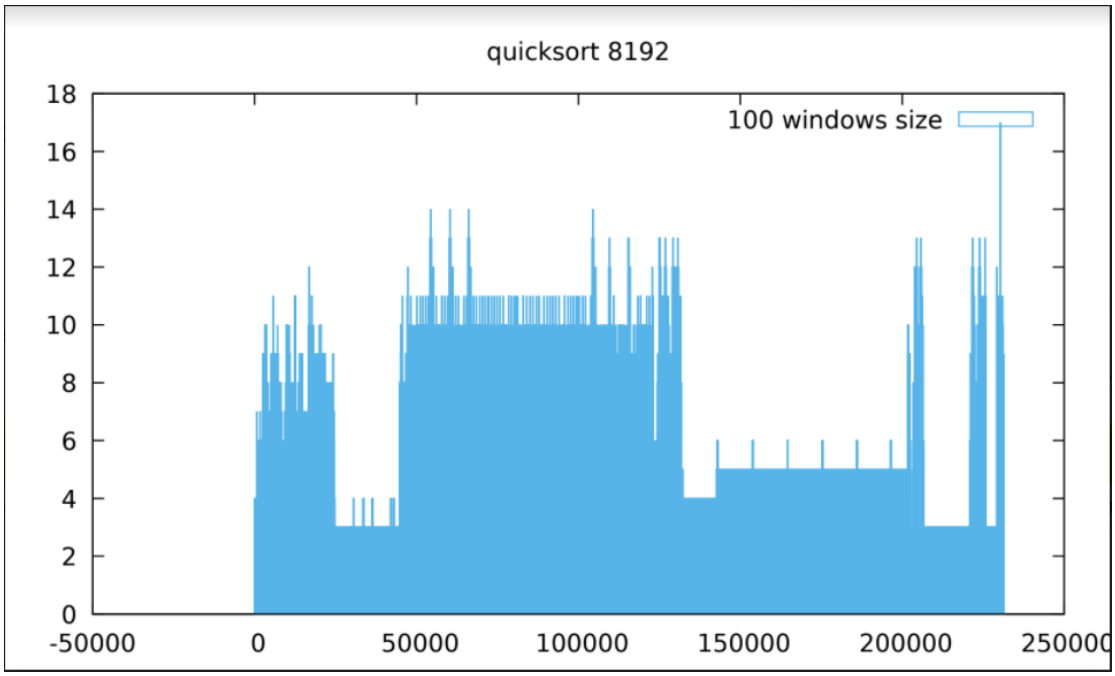


Figure 11
Quicksort input 11, no skip, page size 8192, windows size 1000
Mean: 13.5, variance: 3.6

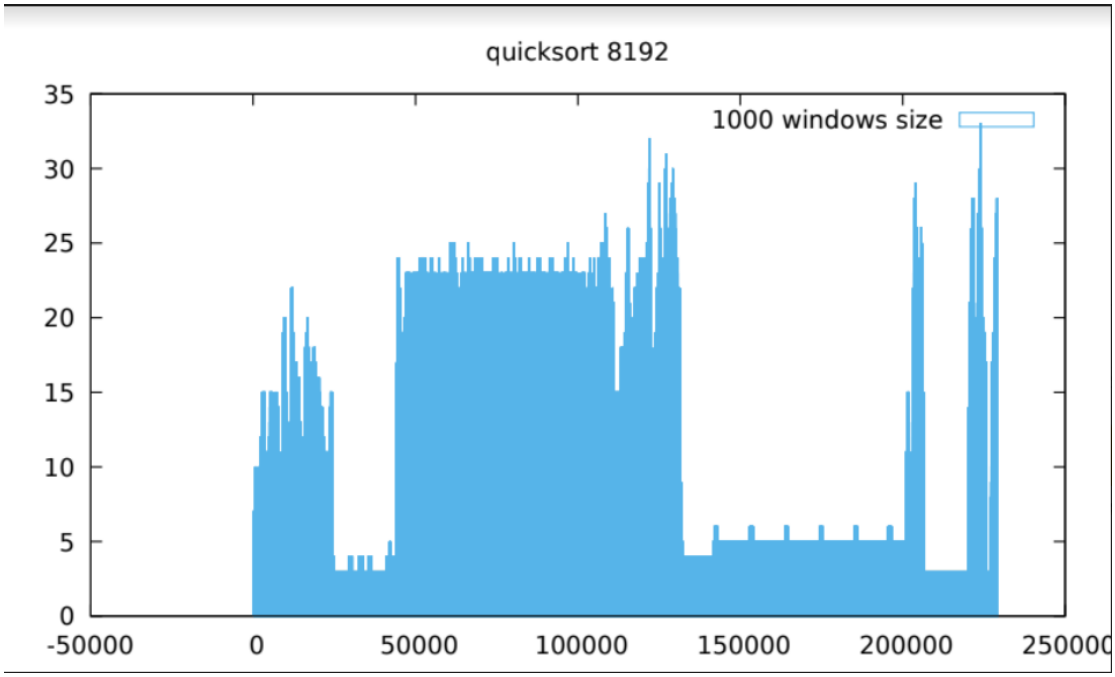


Figure 12
Quicksort input 11, no skip, page size 8192, windows size 10000
Mean: 18.8, variance: 5.3

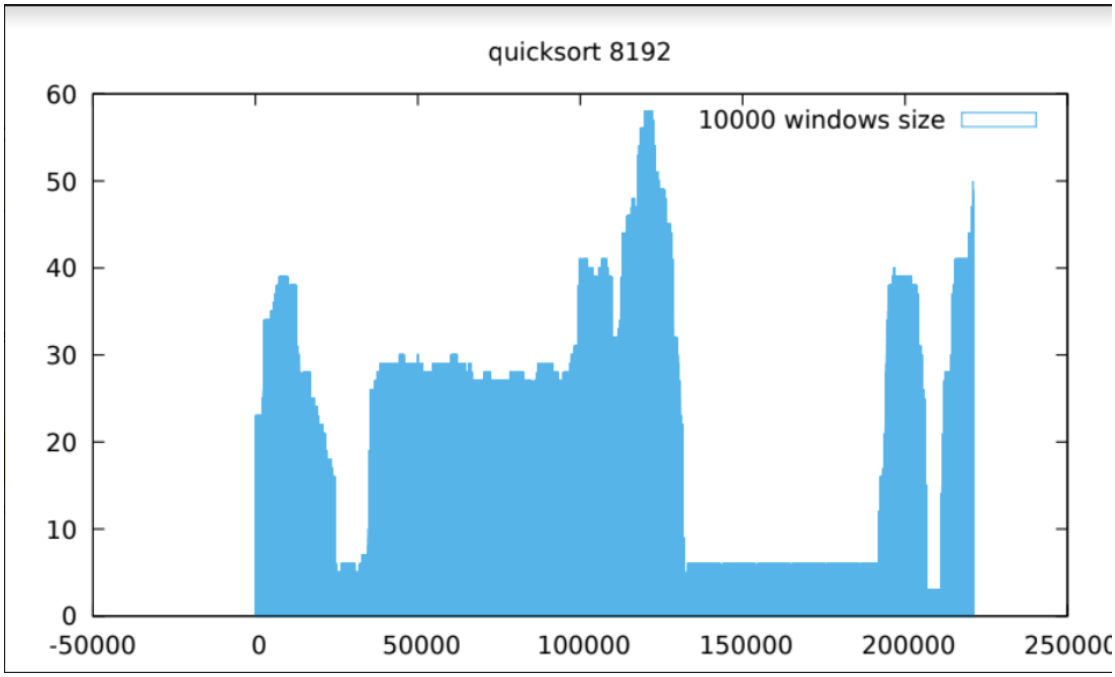


Figure 13
Radixsort input 11, no skip, page size 4096, windows size 100
Mean: 6, variance: 1.7

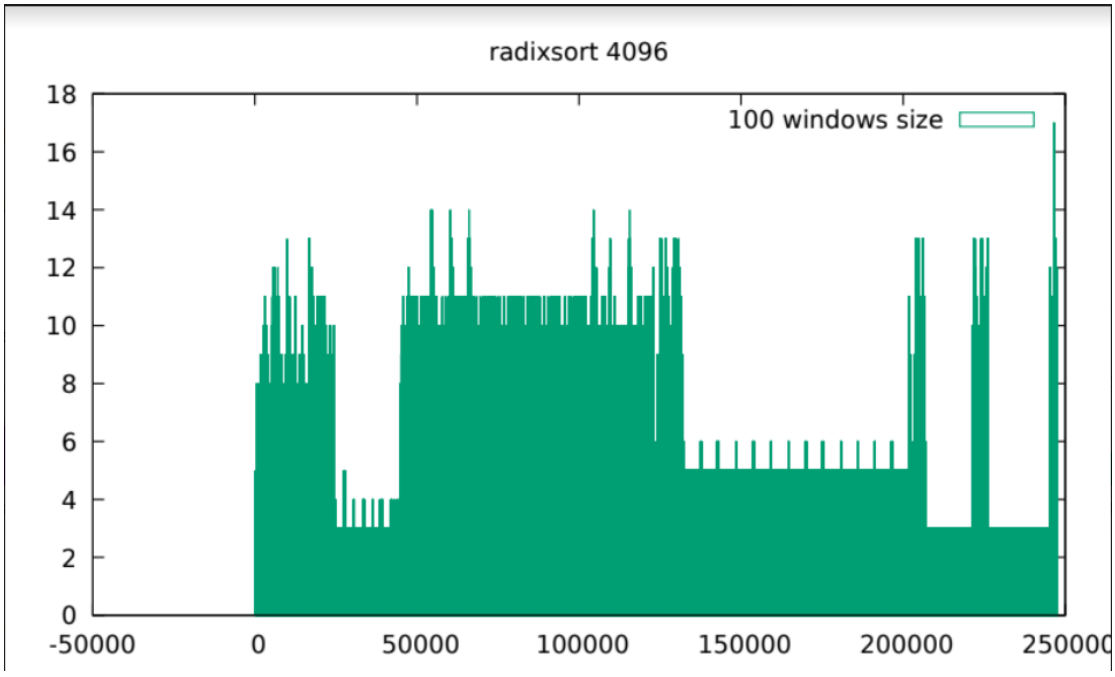


Figure 14
Radixsort input 11, no skip, page size 4096, windows size 1000
Mean: 16.4, variance: 5.3

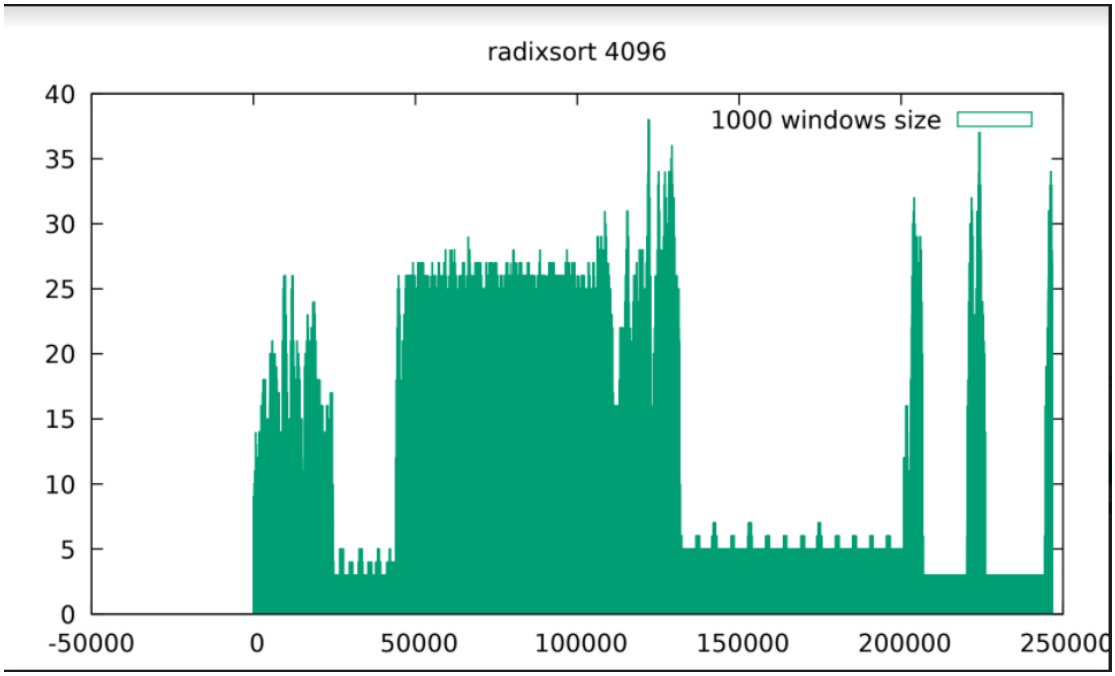


Figure 15
Radixsort input 11, no skip, page size 4096, windows size 10000
Mean: 28, variance: 7.3

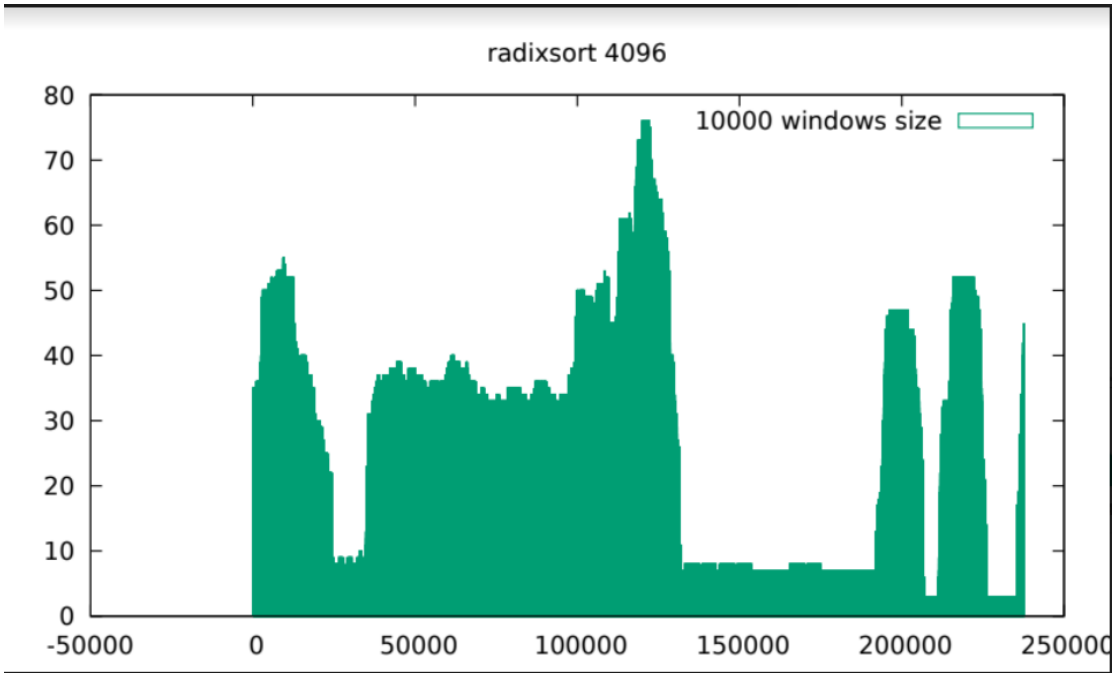


Figure 16
Radixsort input 11, no skip, page size 8192, windows size 100
Mean: 5.7, variance: 2.1

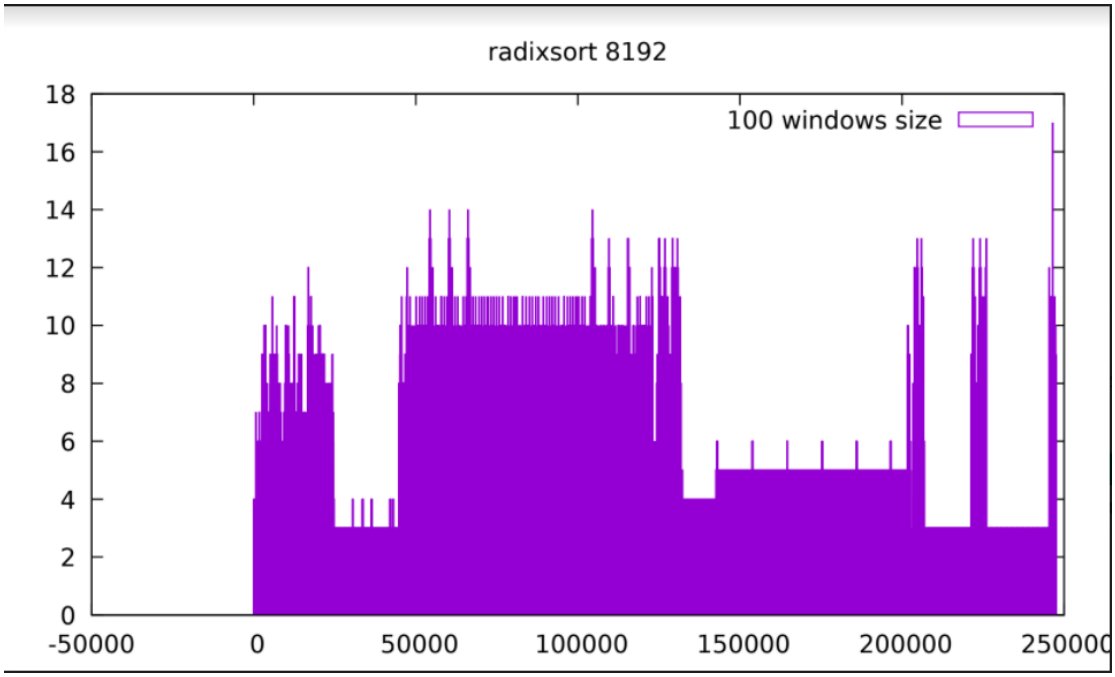


Figure 17
Radixsort input 11, no skip, page size 8192, windows size 1000
Mean: 14.8, variance: 4.5

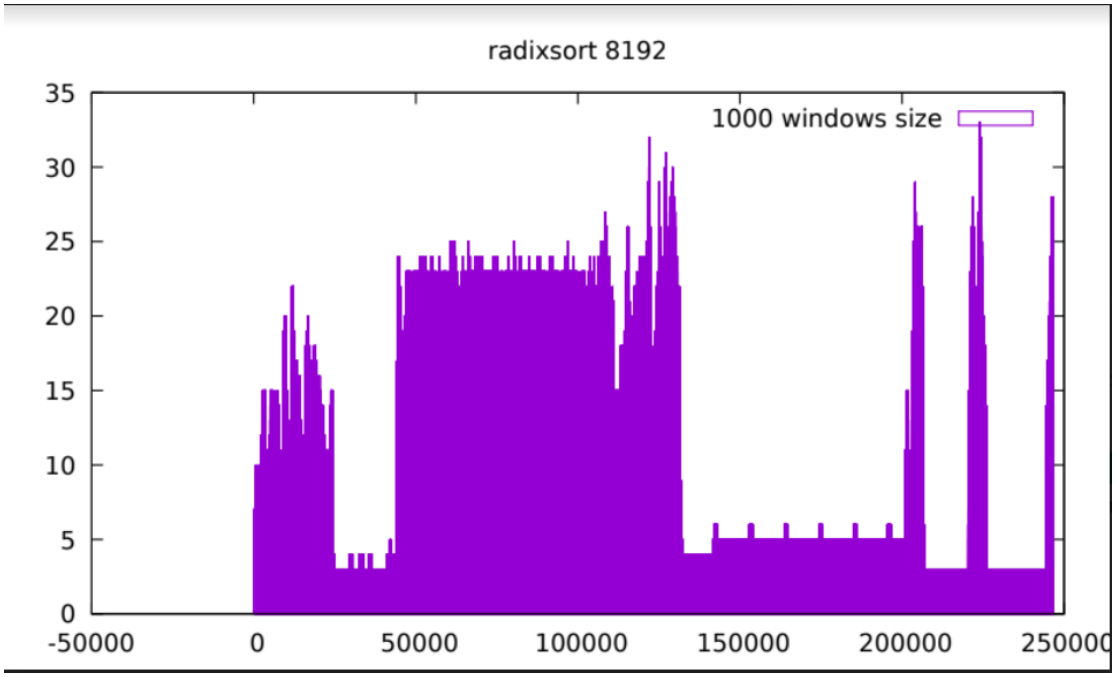


Figure 18
Radixsort input 11, no skip, page size 8192, windows size 10000
Mean: 20, variance: 4.3

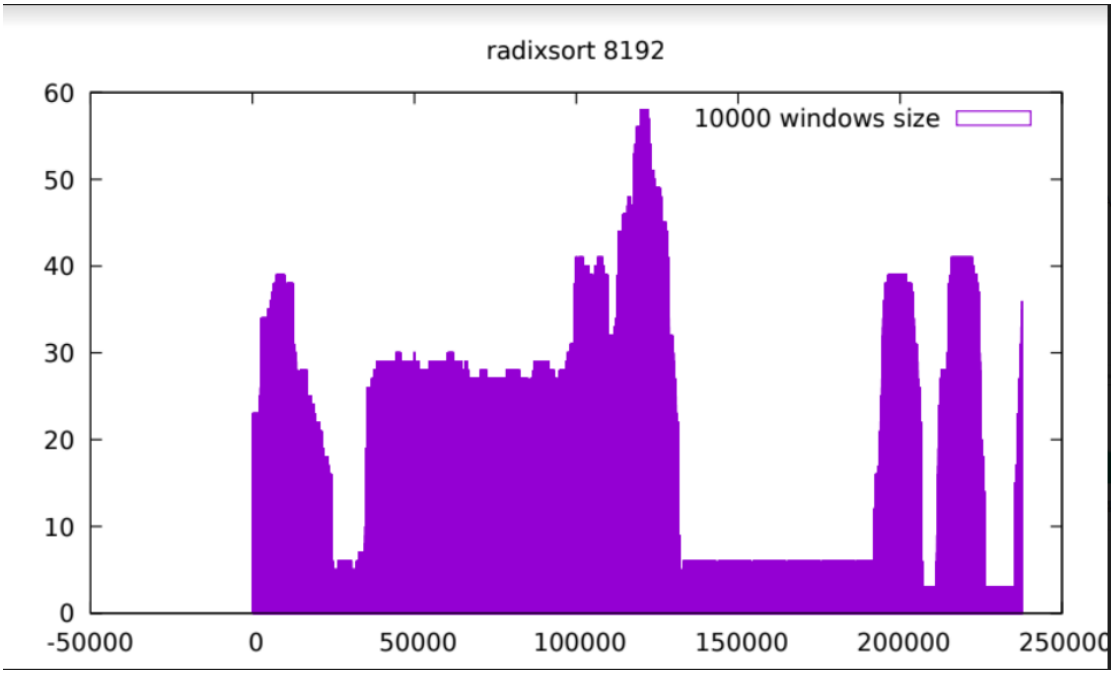


Figure 19
Heapsort input 11, skip 100k, page size 4096, windows size 10000
Mean: 30, variance: 3.5

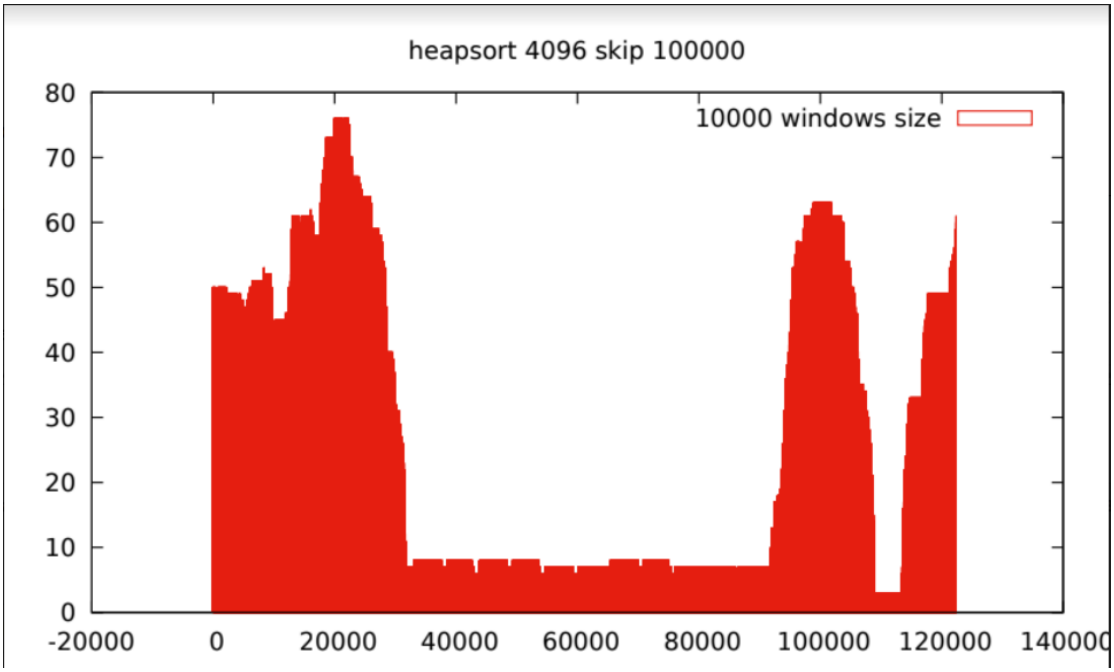


Figure 20
Quicksort input 11, skip 100k, page size 4096, windows size 10000
Mean: 27, variance: 6.2

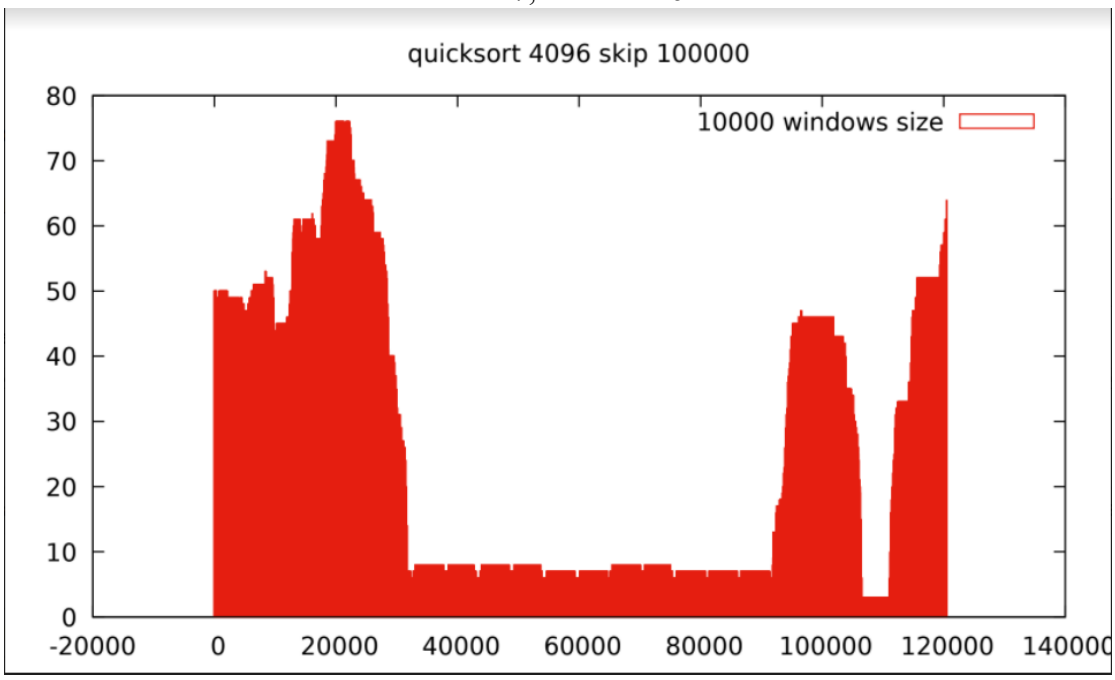


Figure 21
Radixsort input 11, skip 100k, page size 4096, windows size 10000
Mean: 58, variance: 1.5

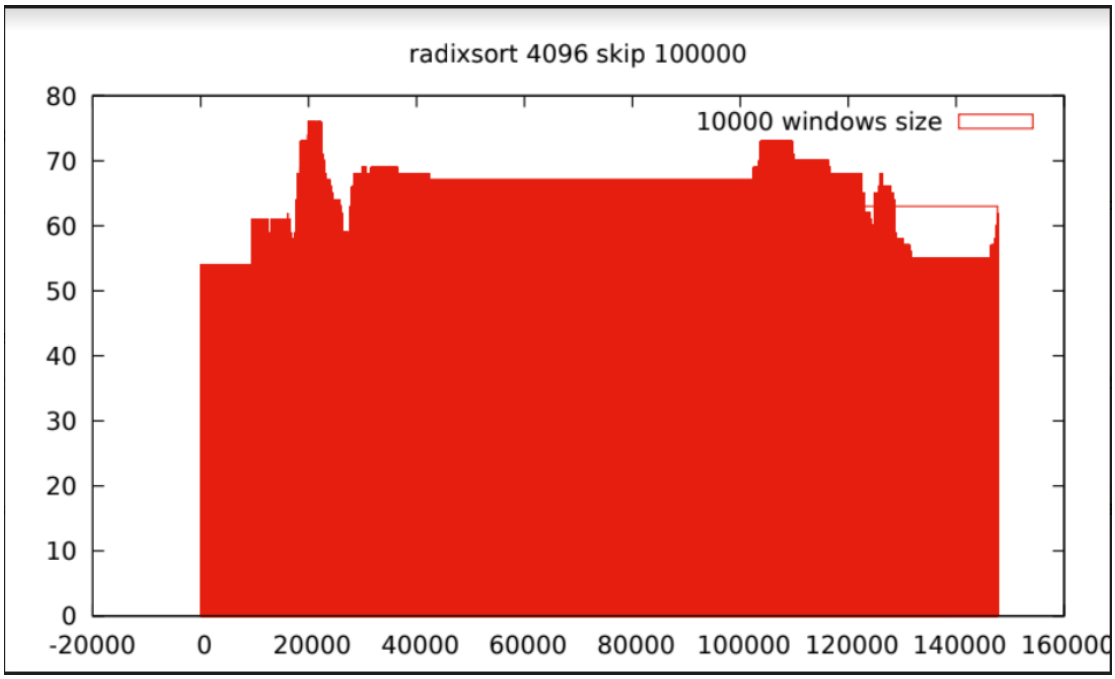


Figure 22
Heapsort input 11, skip 100k, page size 8192, windows size 10000
Mean: 24.3, variance: 5.3

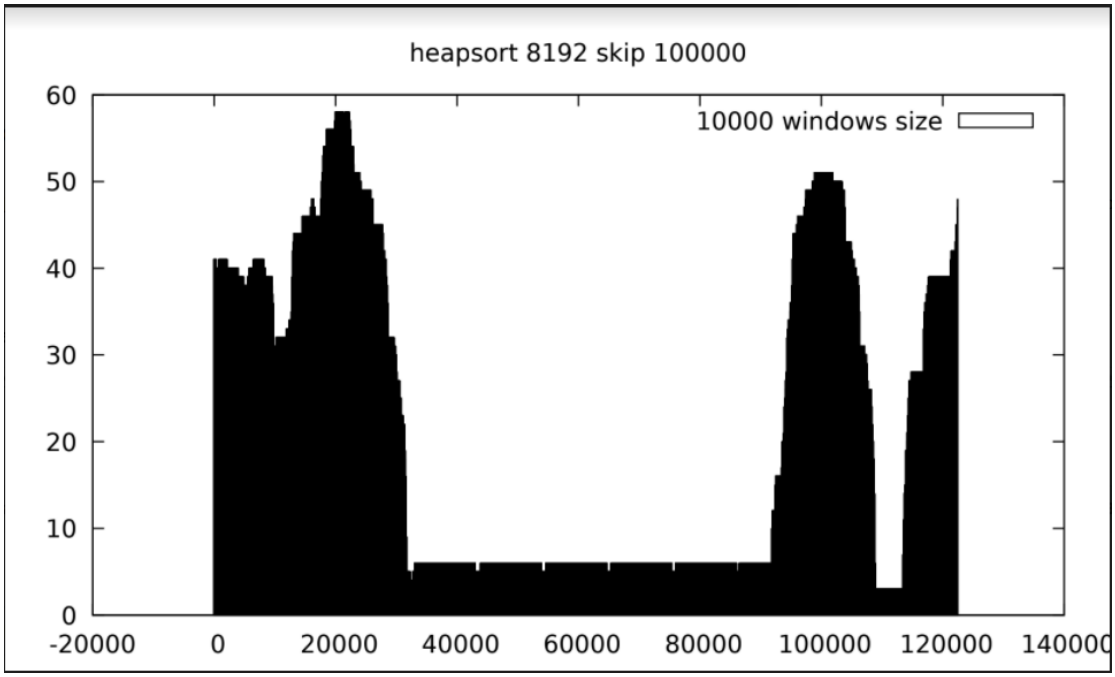


Figure 23
Quicksort input 11, skip 100k, page size 8192, windows size 10000
Mean: 23.6, variance: 5.9

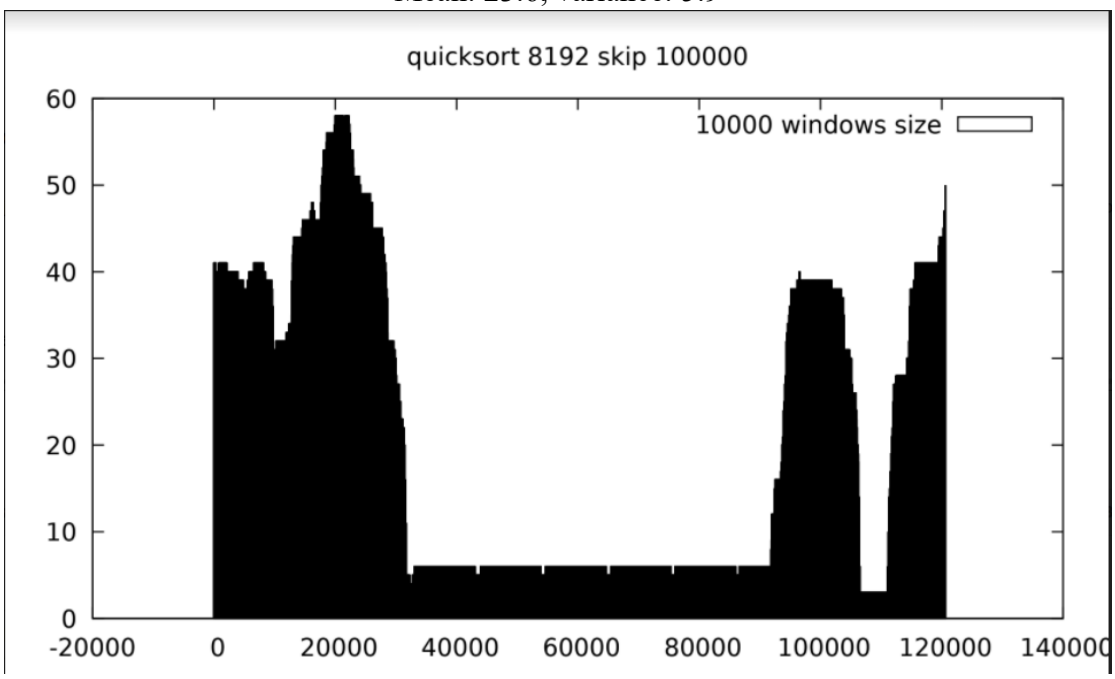
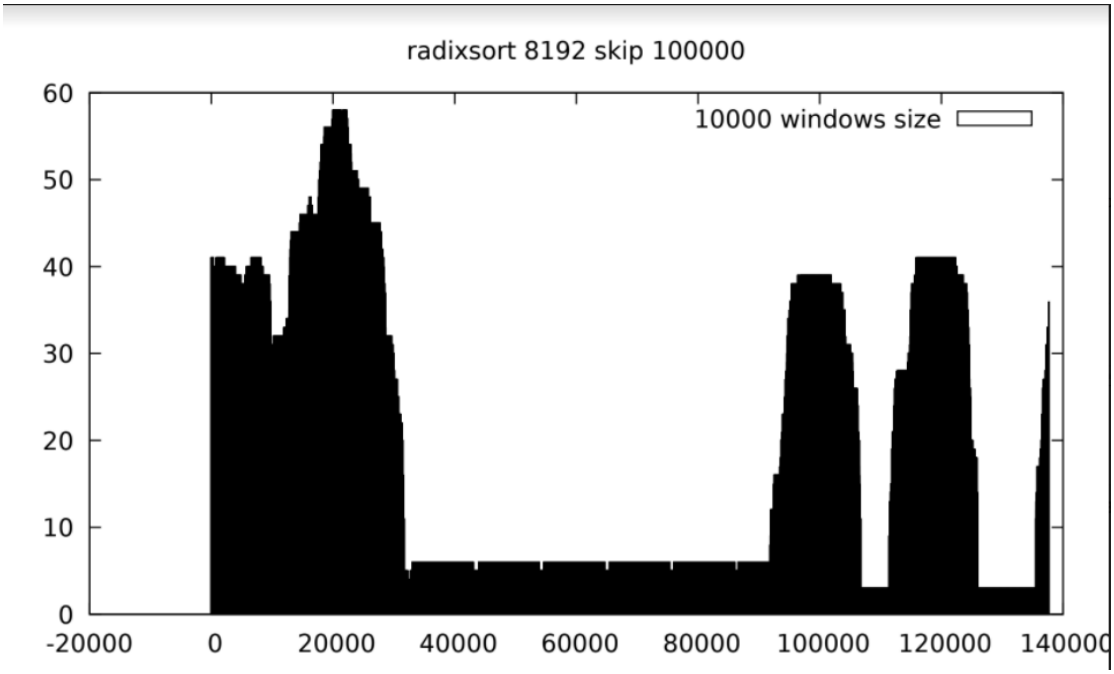


Figure 24
Radixsort input 11, skip 100k, page size 8192, windows size 10000
Mean: 26, variance: 6.3



(note: Figure 25 to 27 will have test program input increase to 100)

Figure 25
Heapsort input 100, no skip, page size 4096, windows size 10000
Mean: 26, variance: 6.3

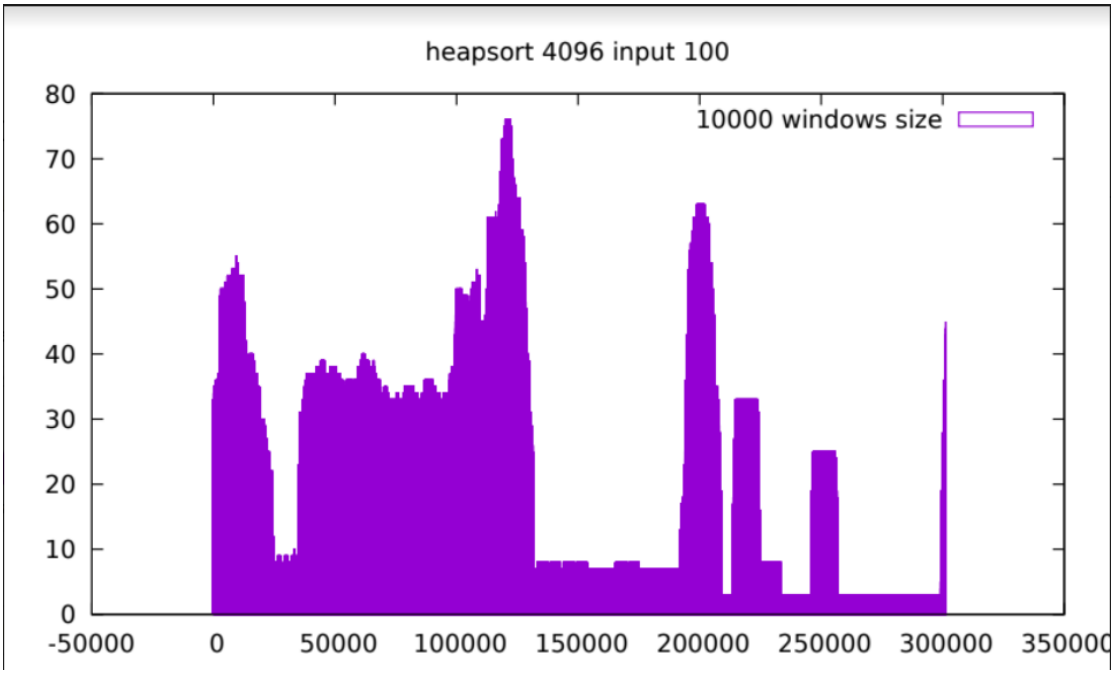


Figure 26
Quicksort input 100, no skip, page size 4096, windows size 10000
Mean: 27.3, variance: 6

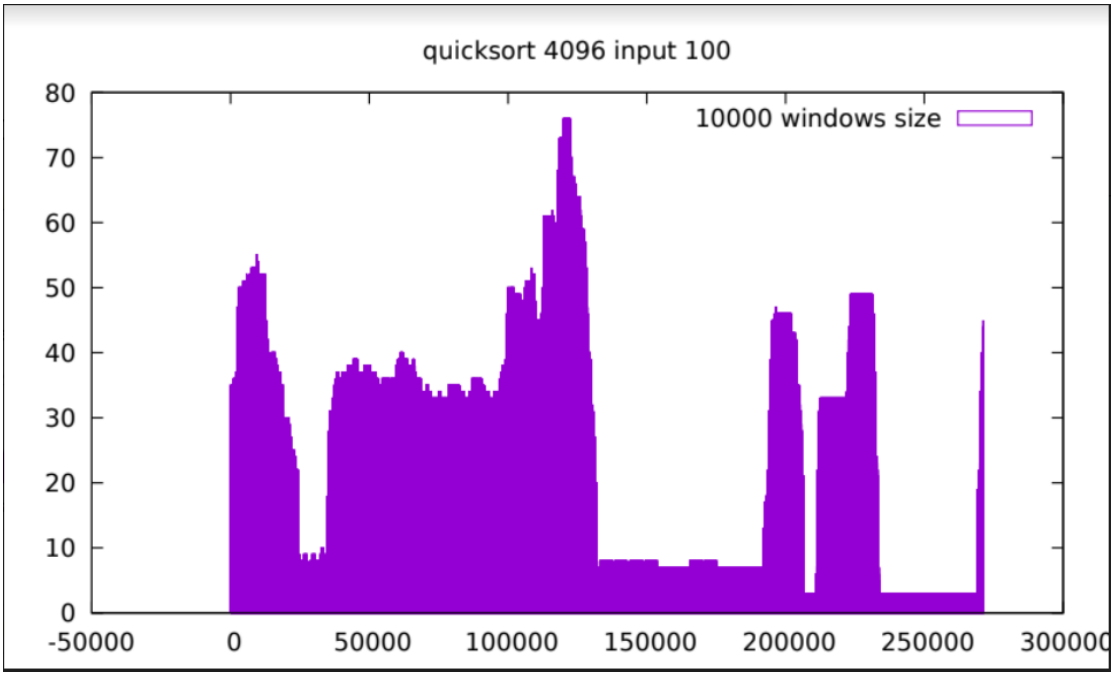
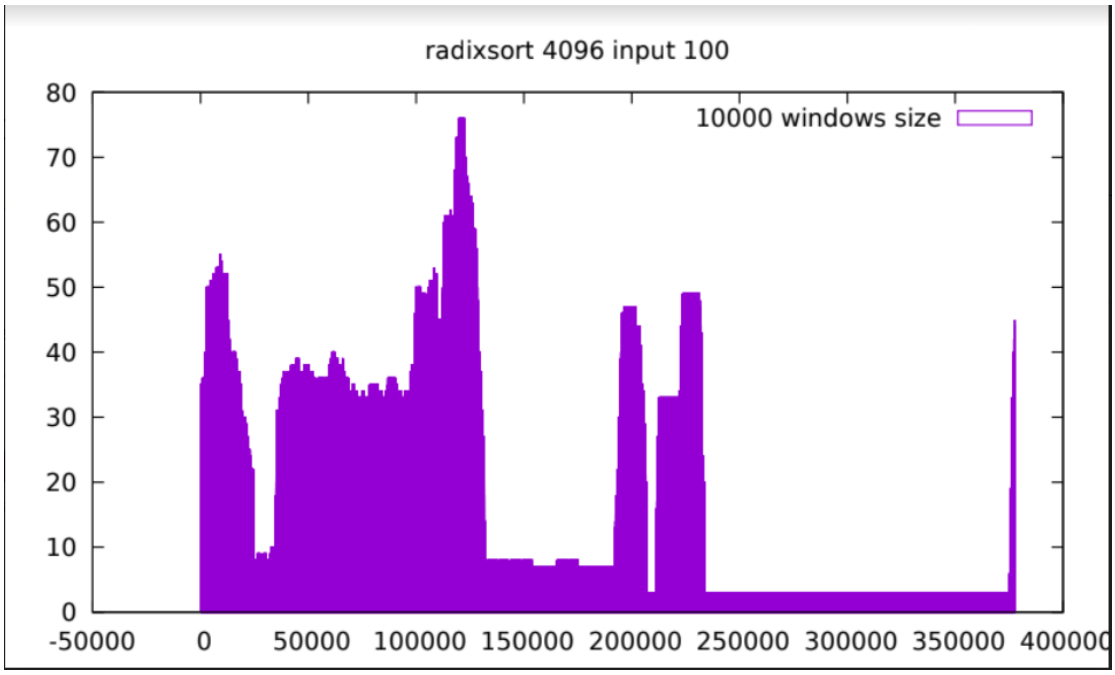


Figure 27
Radixsort input 100, no skip, page size 4096, windows size 10000
Mean: 24, variance: 5.8



Background

Get the output of test program from pipe and check how many unique pages are in the windows set. The windows set will do sliding down to get new lines and keep print current windows set's unique page number.

Use the out put to get histogram and study the performance from the histogram.

Test programs:
--heapsort.c
--quicksort.c
--radixsort.c

Intro to valws379

(Introduce the general flow of how the program work)

Fist the program will check the arguments and store the arguments for the future use.

Second the program will jump to read function to handle the main flow of the program.

Third the program will set up the hash table with malloc memory and a double dimensional array for store what page number had been store in hash table.

Fourth the program will start the loop, the loop will first read valgrind output in the real time and check if that's the line with address and data size. Then will only check the adjacent page and store until the windows size meet. After the size meet will check the adjacent page and store the last line in hash table, report the unique page numbers for the current windows set and save the result into a file. Next, the program will check the oldest page number being stored in hash table by that double dimensional array and remove delete them in both hash table and array (each page number in hash table have a count for store the number of same pages occur. If count greater than one then just minus one, if equal to 1 then remove the page number and minus the unique page number).

Fifth, when the loop is done, free the memory signed for hash table.

Analysis

Similarity:

*These results are all come from the small input of test program, which can not be 100% right, but it fit with the plots shown above.

--The most similar place is all three tests are showing the same frequency at different page size with the same windows size. For example, if you compare Figure 1, 7 and 13, in the beginning all tests showing a high unique page usage and then a quick decrease and raise back contain for a long period of time. Over the halfway, the unique page number are starting to decrease and sometime having a peak.

--The long peak value in the beginning can cause by the argument identifier in each test that checks the argument, store it and pass on to sorting functions. Many temporary pages had been used.

--The second and the longest period of peak value can be cause by the sorting function. When the recursive being use will also use many temporary pages during each calling.

--The short and high peak at the end can be cause by the fprintf used in every test program.

--There are also two rules come with the page number and the windows size.

--The first rule is the larger the page number is the less peak will show. For example, by comparing Figure 1 and 4, Figure 7 and 10, Figure 13 and 16. You can tell the after the page number growth the plot shows less peak which mean the usage of unique page is more stable with less remove. This can cause by when page number growth larger is more likely to have adjacent page and more adjacent page we are using the less temporary page will be use. (The temporary page in smaller page number become stable in large page number might cause by the more adjacent page already signed the page number so the temporary page doesn't get a unique page number). This rule shows the page number and unique page usage are negative linear relation.

--The second rule is when windows size growth the unique page number also growth. For example, by comparing Figure 1 and 2, Figure 7 and 8, Figure 13 and 14. The unique page number of larger windows size plot is higher than be lower windows size plot. Even the unique page usage growth but the layout didn't have a big difference (except the heapsort, will talk about it in the next topic). This result should cause by when the windows size is larger the more unique page are included so the plot are showing more unique page are contain in the same period of time but layout are similar.

Difference:
*These results are all come from the small input of test program, which cannot be 100% right, but it fit with the plots shown above.

--The quicksort and radixsort plot are similar at different page number and windows size combination. For example, Figure 7 and 13, Figure 10 and 16. But the heapsort is likely different with the other sort on the page usage. By compare the Figure 19 to 21 (I strongly felt the Figure 3 and 21 didn't show right, but I tried few times, gnuplot give me the same result so the conclusion might be wrong but its right for the plot I got). The Figure 21 shows a big difference to the other two plots after skip over the first 100k lines. Figure 21 plot showing a continues high unique page usage this can cause by heapsort cover many unique pages without the remove of skipped over lines, it all heaps up to the end.

--Even I didn't get enough data to build a full complete plot to define the difference between each sort, but by the make comparison between Figure 3,9,15 and Figure 25 to 27, there is some trend happening.

--The heapsort seems will have a rising trend and should slowly decrease down after reach a max point. This trend is getting from the plot and how the heapsort was done. The heapsort will take many branches and use many pages to contain the value until the return happens all the temporary pages will be removed. It can be telling by the Figure 1 and 2.

--The radixsort will have a frequently raising up and down based on compare the Figure 15 and 27 and how is the radixsort working like. Every recursion will store in some new values and removed quickly after the work is done. And the plot already shows some frequency happens.

--The quicksort will have a dry constant line after the beginning, this is showing by compare the Figure 9 and 26 with the quick sort's working principle. After the storage in the beginning, its unlikely to have more unique pages being use so the unique pages usage should be low and constant.

Reappear the plot

First will require to use the following command line format:
valgrind --tool=lackey --trace-mem=yes ./radixsort <input value>
2>&1|./valws379 <-s skipize> <page number><windows size>

The input value, skipize, page number and windows size had all give in the figure's pages above each plot.

Second to use gnuplot:
set terminal pdf (this part should be png)
#set up a name for the output pdf
set output "pageSize 4096 10000 input 100 radixsort.pdf"
#set up the title for the plot graph
set title "radixsort 4096 input 100"
#sign r100 to be the file to read data from, output as histogram with type 9 color and give a sign to the data
plot "r100" with boxes lt 9 title "10000 windows size"