Maggie McComas

Based on the outputs of my programs, as can be seen in the figures below, the median of medians and quicksort algorithms perform better than the randomized selection algorithm on smaller arrays (Figures 1 and 2). As the size of the array increases, randomized selection finds the median faster than the median of medians algorithm (Figures 3, 4, and 6). Both randomized selection and the median of medians algorithm noticeably outperform quicksort on larger arrays though (Figures 3, 4, 5, and 6). The best-case performance for quicksort is O($nlogn$). The worst-case performance for quicksort is O(n^2). The best-case performance for the median of medians algorithm is O(n) and its worst-case performance is the same. The best-case performance for the randomized selection algorithm is O(n), and its worst-case is O(n^2). Based on their best and worst-case performances along with the output from my programs, the median of medians algorithm performs the best. The median of medians algorithm consistently performed very well in my programs. Although it did not have the fastest time for the larger arrays, it was not that much slower than the randomized selection algorithm. If done many more times, the time it takes for the median of medians algorithm to find the median of large arrays will stay fairly consistent, but the time it takes the randomized selection algorithm to find the median of large arrays would most likely vary as can be seen by its worst-case performance of O(n^2). In addition, the median of medians algorithm performed faster than quicksort for one of the smaller arrays and slower by 0.000002 seconds for the other. This is not noticeable at all to most users. If done many more times, the time it takes for the median of medians algorithm to find the median of small arrays will stay fairly consistent, but the time it takes quicksort to find the median of small arrays would most likely vary as can be seen by its worst-case performance of O(n^2). Therefore, the median of medians algorithm has the best performance because it will consistently perform very well on arrays no matter their size. Its worst-case performance is also better than the worst-case performance of quicksort and randomized selection.

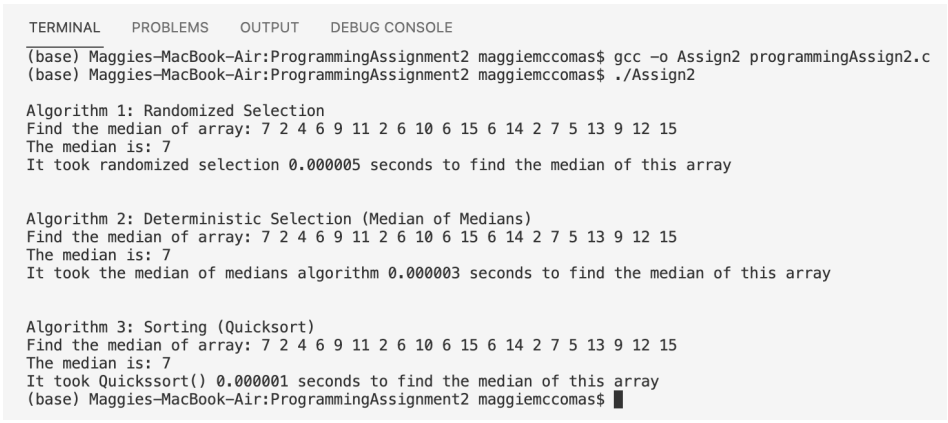

Figure 1. The output for each algorithm using the given array as well as how long it took each algorithm to find the median of this array.

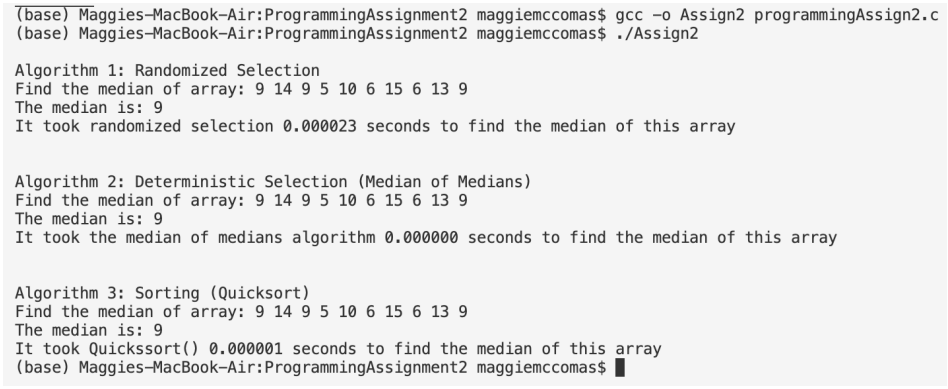

Figure 2. The output for each algorithm using the practice array to make sure they all output 9. As well as how long it took each algorithm to find the median of this array.

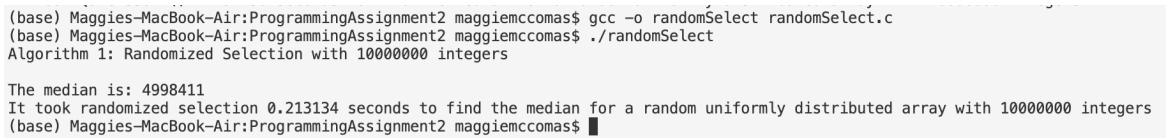

Figure 3. The output for the randomized selection algorithm with the given input of a randomized array with 10000000 elements. Along with how long it took to find the median.

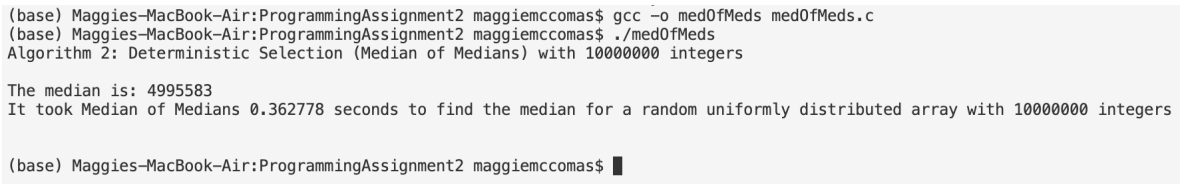

Figure 4. The output for the median of medians algorithm with the given input of a randomized array with 10000000 elements. Along with how long it took to find the median.

```
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$ gcc -o quickSort quickSort.c
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$ ./quickSort
Algorithm 3: Sorting (Quicksort) with 10000000 integers

The median is: 4993460
It took Quickssort() 1.791750 seconds to find the median for a random uniformly distributed array with 10000000 integers
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$
```

Figure 5. The output for the quicksort algorithm with the given input of a randomized array with 10000000 elements. Along with how long it took to find the median.

```
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$ gcc -o who who.c
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$ ./who
Algorithm 3: Sorting (Quicksort) with 100000000 integers

The median is: 48868914
It took Quickssort() 22.876775 seconds to find the median for a random uniformly distributed array with 100000000 integers
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$ gcc -o who who.c
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$ ./who
Algorithm 2: Deterministic Selection (Median of Medians) with 100000000 integers

The median is: 48874052
It took Median of Medians 3.780198 seconds to find the median for a random uniformly distributed array with 100000000 integers


(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$ gcc -o who who.c
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$ ./who
Algorithm 1: Randomized Selection with 10000000 integers

The median is: 48873288
It took randomized selection 2.833954 seconds to find the median for a random uniformly distributed array with 100000000 integers
(base) Maggies-MacBook-Air:ProgrammingAssignment2 maggiemccomas$
```

Figure 6. The outputs for the three algorithms with the given input of a randomized array with 100000000 elements. Along with how long it took to find the median.