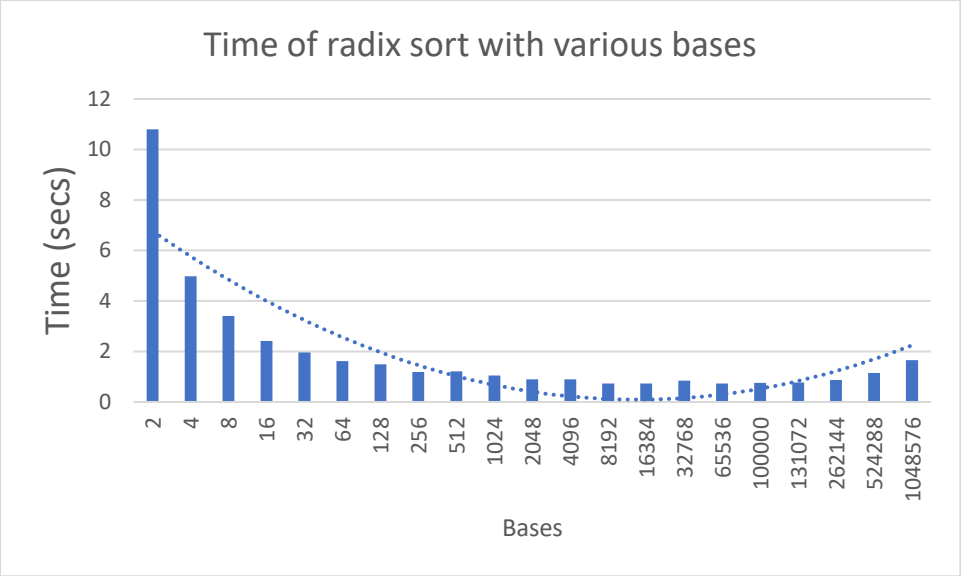
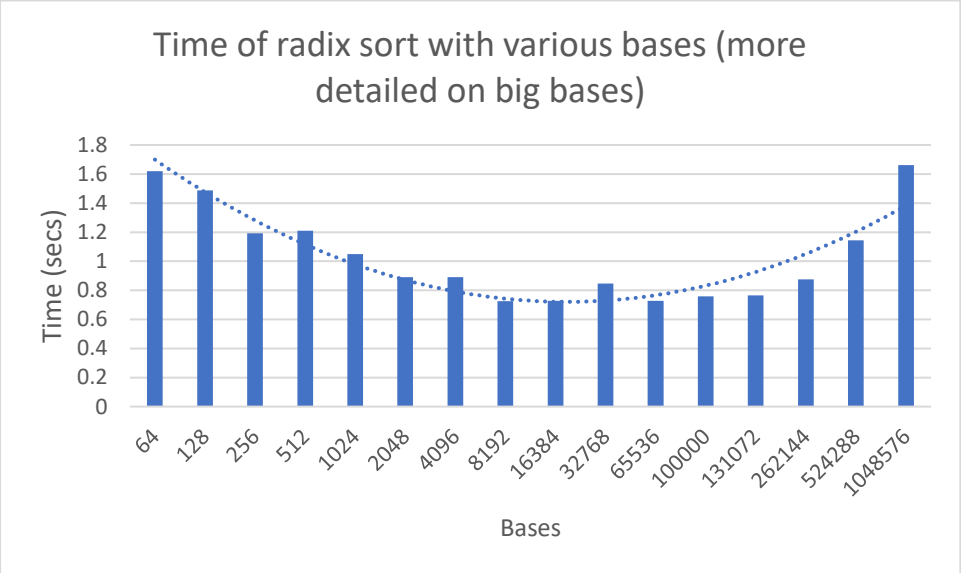
Task2

I have obtained mainly the bases that are powers of 2. This is because the computer is built based on the power of 2. Additionally, we can save a constant factor and we can access groups of bits using power of 2 bases by using bit operators, meaning the number of count sorts are halved over time again and again when base increases.





I have presented two graphs here, the first one with all the bases I picked and the second graph with more detailed discrepancy between the timing of larger bases. We can see that there is a polynomial dotted line here, indicating that the time is approximately the lowest when the base is around 2^14.

We need to look at the big O-notation here for radix sort, which is O((n + b)*m), where n is the total number of integers in the input list, b is the base and m is the number of digits in the largest number in the input list, when represented in base b. m here is calculated by $\log_b M$, where M is the number of largest value in its original base. Additionally, we require count sort to make radix sort efficient. This means, we have to run count sort $\log_b M$ times to sort a number size of M. Each count sort requires O(n + b). So if we increase b, the number of count sorts goes down ($\log_b M$ will be smaller if

the base is large), but the cost of each count sort goes up (n + b will be greater as b is increased). If we decrease b, vice versa.

So, we would want to find a good balance, where the base we use can reduce both the cost of count sort and numbers of times of running count sort. If we let b = N, the total cost would become $O(\log_n M*n)$ as O(2*n) -> O(n). Moreover, if M is O(n^c), given c is a constant, the total cost will become O(CN), which becomes O(n). However, this is only a theoretical big O-notation.

In the graphs, we can see that at the start (left hand side) the bases are small, which made the $\log_b M$ to be a big number and there had to be many count sorts to be done. As we approach to the bases in the middle of the graph (b = 2^14), there seems to be a better balance between the cost of each count sort and number of count sorts. Therefore, the time decreased dramatically. We can also see the minimum of the polynomial graph resting just on top of base of 2^14. Notably, since the list of random input number has a total size of 100,000, I have included the base of 100,000 in the graph to verify if it can reach O(n) times. Having base of n may give you one of the best times, but there are other bases that can sort faster, indicating that the theory might not work all the time in practical work. Close to the right-hand side, we have very large bases with last one being 2^21. We can see that the time has significant increased. This is most likely due the cost of count sort being increased greatly. Although the number of count sort required is reduced, the cost of each count sort is increased by a large amount due to O(n + b), to the point that b is 10 times greater than n. Most power of 2's that is approximately close to n is a good base be practically fast and theoretically fast.