**Introduction and Methods:**

The aim of this programming assignment was to implement and compare Heapsort algorithms that ran with arbitrary branching factors (as opposed to having a branching factor of just 2). Heapsorts with different branching factors were tested on unsorted input arrays to determine firstly, which factor of Heapsort ran fastest for a given value of ‘N’, and secondly, whether the specific branching factor for Heapsort that ran in the shortest time depended upon the size of the input (‘N’).

A method with the signature: void check (int [] a, int [] b) was used to first compare the results of Insertion Sort (which was checked to be working correctly using the method with the signature: void print (int [] a)) and each Heapsort of branching factors from 2 to 10 respectively on the same input array for values of ‘N’ from 101 to 105 in order to ensure that the Heapsort algorithm worked correctly for all branch values. Then the Insertion Sort and the “checker” method were commented out in order to test higher values of ‘N’. For this investigation, nine different values of branching factors, from 2 to 10 inclusive, were tested. To determine which branching factor ran fastest for a given value of ‘N’, a value of ‘N’ that was neither too small nor too large was chosen (N = 107) and each of the Heapsorts (which had a distinct individual branching factor) was run on an input array of size ‘N’. The time taken for each to completely run was recorded and tabulated. To determine whether the “best” branching factor depended on input size, values of ‘N’ were varied from 103 to 108 inclusive and each Heapsort was run on an input array of each size. Respective running times for each Heapsort were then recorded and tabulated.

For each reading, in both of the experimental questions, five trials were taken and then averaged in order to minimize errors. Moreover, each Heapsort was given the same input array for the same value of ‘N’ in order to standardize the input array for one trial. This was effected by creating copies of the initially randomly generated array of size ‘N’. Care was taken to ensure correctness of each Heapsort and that recorded times corresponded to times taken to **only** sort (and not “print” or “check”, for example, the arrays).

**Results and Discussion:**

From the series of trials carried out in the first part of the investigation, it was concluded that a branch factor of 5 gave the fastest average running time, followed by a branch factor of 4. Moreover, running times decreased from branch factor = 2 to branch factor = 5 only to increase for higher branch factors (Table 1).

From the second part of the investigation, the **overall** best branch factor remained 5 for values of inputs from ‘N’ = 103 to 107, but decreased to 4 for ‘N’ = 108; henceforth, it was concluded that the best branching factor decreases for large values of ‘N’ (>107) (Tables 2 through to 10). The blip in the performance of the Heapsort with a branch factor of 5 (ie, the value of ‘N’ for which it was not the fastest) occurred at ‘N’ = 105 (at which the best branch factor was 7). This blip could be explained by the inherent inaccuracy in timing (as discussed later). Heapsort with the branch factor of 4 approached the speed of sorting of Heapsort with the branch factor of 5 for values of ‘N’ lower than 108, eventually beating it for ‘N’=108. For values of ‘N’ <108, the branch factor of 2 gave the slowest running time of all the Heapsort algorithms, followed by the branch factor of 10. However, for ‘N’ = 108, a branch factor of 10 gave the fourth fastest running time of all Heapsorts, thereby leading us to conclude that at higher values of ‘N’, Heapsort with the highest branch factor of 10 ran relatively much faster compared to its running time at lower values of ‘N’.

The reason why the best branch factor is an intermediate and not an extreme can be attributed to two major trade-offs. With an increasing branch factor, for the same value of ‘N’, the heap has a height that is smaller than the height of a heap with a smaller branching factor by a constant value. This means there are a smaller number of levels for the tree, but a greater number of elements for each level (except the first). This allows for firstly, a reduced number of calls to heapify in the recursive method (because of a decreased height), and secondly, the program to access a larger number of continuous element indices (a greater number of “fast” indexes). These factors speed up the running time of the algorithm. However, the trade-off with an increasing branch factor is the increased amount of work (because a greater number of children need to be compared) at each call to heapify, and a larger number of “slow” indexes to be skipped over. These trade-offs favor an intermediate value for the optimum branch factor, which was found to be 5. Moreover, these trade-offs affect each branching factor uniquely when the value of ‘N’ is increased, thereby causing different effects for each algorithm (for example the rapid relative improvement in the performance of Heapsort with a branching factor of 10 when ‘N’ is increased from 107 to 108  as compared to other Heapsorts).

From the investigation, it was also seen that the best value of the branching factor depended upon the actual values in the input array (as opposed to just the size of the input array). This can be seen by how Heapsort with a branch factor of 4 beats Heapsort with a branch factor of 5 for two trials (labelled in red in Table 1) for ‘N’ = 107. This can be justified by the argument that each branching factor causes a unique arrangement of the elements of the input array in terms of the tree created, that will, in turn affect where an element of a certain value is placed in the tree. This would ultimately affect the running time of Heapsort, as there could be a differing number of recursive calls to heapify for each Heapsort. In other words, the values in the input array that cause the worst case for one branching factor may not cause the worst case for another branching factor.

A considerable uncertainty in the investigation was the time logging technique. Due to inherent inaccuracies, the time logged was to a maximum precision of 2 decimal places. One example of the inconvenience this caused was in the second part of the investigation, when recording times for Heapsort with a factor of 5 and 7 at ‘N’ = 105 respectively. On average, Heapsort with the factor of 7 is “faster,” but when comparing the running times for individual trials, both Heapsorts take exactly the same time for 4 out of 5 trials. Due to the inability to express higher levels of decimal places, Heapsort with a branch factor of 5 runs twice as slow as the Heapsort with a branch factor of 7 in trial 1 (labelled in green in tables 5 and 7), which causes Heapsort with a branch factor of 7 to be the “fastest” at a value of ‘N’ = 105, even though much of the accuracy has been lost due to premature rounding off.

**Appendix:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Branch factor | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 2 | 4.200 | 4.160 | 2.920 | 2.980 | 3.010 | 3.454 |
| 3 | 2.900 | 2.860 | 2.200 | 2.390 | 2.260 | 2.522 |
| 4 | 2.670 | 2.670 | 2.090 | 2.210 | 2.150 | 2.358 |
| 5 | 2.520 | 2.240 | 2.100 | 2.220 | 2.120 | 2.240 |
| 6 | 2.640 | 2.500 | 2.160 | 2.250 | 2.400 | 2.390 |
| 7 | 2.690 | 2.610 | 2.210 | 2.300 | 2.580 | 2.478 |
| 8 | 2.850 | 2.700 | 2.380 | 2.440 | 2.740 | 2.622 |
| 9 | 2.900 | 2.780 | 2.410 | 2.500 | 2.750 | 2.668 |
| 10 | 3.030 | 2.910 | 2.560 | 2.630 | 2.900 | 2.806 |

**Table 1 (N= 107).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.010 | 0.010 | 0.010 | 0.000 | 0.006 |
| 104 | 0.010 | 0.000 | 0.010 | 0.010 | 0.010 | 0.008 |
| 105 | 0.020 | 0.020 | 0.020 | 0.020 | 0.030 | 0.022 |
| 106 | 0.210 | 0.210 | 0.210 | 0.210 | 0.210 | 0.210 |
| 107 | 4.440 | 4.250 | 4.240 | 2.960 | 4.060 | 3.990 |
| 108 | 71.660 | 97.160 | 82.970 | 65.110 | 78.120 | 79.004 |

**Table 2 (Branch factor = 2).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 104 | 0.000 | 0.000 | 0.000 | 0.000 | 0.010 | 0.002 |
| 105 | 0.010 | 0.020 | 0.010 | 0.010 | 0.010 | 0.012 |
| 106 | 0.160 | 0.170 | 0.150 | 0.160 | 0.160 | 0.160 |
| 107 | 2.270 | 2.840 | 2.930 | 2.230 | 2.800 | 2.614 |
| 108 | 58.42 | 50.92 | 56.92 | 57.14 | 60.85 | 56.85 |

**Table 3 (Branch factor = 3).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 104 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 105 | 0.010 | 0.010 | 0.010 | 0.010 | 0.020 | 0.012 |
| 106 | 0.160 | 0.150 | 0.160 | 0.150 | 0.160 | 0.156 |
| 107 | 2.620 | 2.760 | 2.720 | 2.130 | 2.650 | 2.576 |
| 108 | 46.73 | 47.06 | 46.26 | 45.27 | 42.88 | 45.64 |

**Table 4 (Branch factor = 4).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 104 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 105 | 0.020 | 0.010 | 0.010 | 0.010 | 0.010 | 0.012 |
| 106 | 0.150 | 0.160 | 0.160 | 0.160 | 0.150 | 0.156 |
| 107 | 2.730 | 2.540 | 2.550 | 2.090 | 2.520 | 2.486 |
| 108 | 48.910 | 51.610 | 50.400 | 43.210 | 47.860 | 48.398 |

**Table 5 (Branch factor = 5).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 104 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 105 | 0.010 | 0.010 | 0.020 | 0.020 | 0.010 | 0.014 |
| 106 | 0.170 | 0.170 | 0.160 | 0.160 | 0.170 | 0.166 |
| 107 | 2.710 | 2.590 | 2.570 | 2.160 | 2.600 | 2.526 |
| 108 | 82.910 | 49.550 | 48.050 | 47.380 | 47.250 | 55.028 |

**Table 6 (Branch factor = 6).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 104 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 105 | 0.010 | 0.010 | 0.010 | 0.010 | 0.010 | 0.010 |
| 106 | 0.170 | 0.170 | 0.170 | 0.160 | 0.170 | 0.168 |
| 107 | 2.770 | 2.690 | 2.600 | 2.230 | 2.660 | 2.590 |
| 108 | 76.700 | 47.160 | 48.640 | 48.450 | 47.090 | 53.608 |

**Table 7 (Branch factor = 7).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 104 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 105 | 0.020 | 0.020 | 0.010 | 0.010 | 0.020 | 0.016 |
| 106 | 0.190 | 0.180 | 0.180 | 0.180 | 0.190 | 0.184 |
| 107 | 2.810 | 2.880 | 2.720 | 2.360 | 2.800 | 2.714 |
| 108 | 49.440 | 50.200 | 50.740 | 50.990 | 48.860 | 50.046 |

**Table 8 (Branch factor = 8).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 104 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 105 | 0.010 | 0.010 | 0.020 | 0.020 | 0.010 | 0.014 |
| 106 | 0.190 | 0.190 | 0.190 | 0.190 | 0.190 | 0.190 |
| 107 | 2.860 | 2.830 | 2.670 | 2.410 | 2.830 | 2.720 |
| 108 | 52.390 | 56.350 | 51.960 | 57.920 | 53.070 | 54.338 |

**Table 9 (Branch factor = 9).**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘n’ | t1/s | t2/s | t3/s | t4/s | t5/s | <t>/s |
| 103 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 104 | 0.000 | 0.000 | 0.000 | 0.010 | 0.000 | 0.002 |
| 105 | 0.020 | 0.020 | 0.010 | 0.010 | 0.020 | 0.016 |
| 106 | 0.200 | 0.200 | 0.200 | 0.200 | 0.200 | 0.200 |
| 107 | 3.030 | 2.950 | 2.960 | 2.550 | 3.010 | 2.900 |
| 108 | 54.920 | 45.080 | 56.050 | 45.540 | 55.540 | 51.426 |

**Table 10 (Branch factor = 10).**