1. */\**
2. *\* C++ Program to Implement Heap Sort*
3. *\*/*
4. #include <iostream>
5. #include <conio.h>
6. using namespace std;
7. void max\_heapify(int \*a, int i, int n)
8. {
9. int j, temp;
10. temp = a[i];
11. j = 2\*i;
12. while (j <= n)
13. {
14. if (j < n && a[j+1] > a[j])
15. j = j+1;
16. if (temp > a[j])
17. break;
18. else if (temp <= a[j])
19. {
20. a[j/2] = a[j];
21. j = 2\*j;
22. }
23. }
24. a[j/2] = temp;
25. return;
26. }
27. void heapsort(int \*a, int n)
28. {
29. int i, temp;
30. for (i = n; i >= 2; i--)
31. {
32. temp = a[i];
33. a[i] = a[1];
34. a[1] = temp;
35. max\_heapify(a, 1, i - 1);
36. }
37. }
38. void build\_maxheap(int \*a, int n)
39. {
40. int i;
41. for(i = n/2; i >= 1; i--)
42. {
43. max\_heapify(a, i, n);
44. }
45. }
46. int main()
47. {
48. int n, i, x;
49. cout<<"enter no of elements of array**\n**";
50. cin>>n;
51. int a[20];
52. for (i = 1; i <= n; i++)
53. {
54. cout<<"enter element"<<(i)<<endl;
55. cin>>a[i];
56. }
57. build\_maxheap(a,n);
58. heapsort(a, n);
59. cout<<"sorted output**\n**";
60. for (i = 1; i <= n; i++)
61. {
62. cout<<a[i]<<endl;
63. }
64. getch();

Using multi threads (at least 4 cores)

Create array for the sorting

Clear out data (free memory)--> re-initialize pointers

Need destructor

ASCII values are only ones considered

Higher performance over memory constraints

Read data should clear out data

Look out for different data sets

Insertion sort better than quicksort for small data

Hybrid: QuickSort-> insertion sort (9 or less)

At beginning of read data should Clear vector

Macros

* Then do threads

Compile using C++ 11

Comment Sorting algorithm (can cite sorting algorithm)

Sort by: length then alphabetize

Length:

1. Overhead with strlen()
2. Instead do pre-fixed length on char\*

Alphabetize:

Quicksort:

1. Quicksort in place

Strengths and Weaknesses of Data Sets

1. Quicksort:

2. Bubble Sort:

- small sets of data

To compile g++ -std=c++11 –stdlib=libc++

CONFIG += C++11

\*\*\*\*\*\*\*\*\*\*\*\*\*Tell if need compile directives\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main method:

Sorting competition mySorter = SortingCompetition(file)

Unsigned long long avg = 0;

Unsinged long long total

For(unsigned int i…)

//starts chrono

Auto start = chrono::system\_clock::now();

Sort data

Auto stop = chrono::system\_clock::now();

Auto diff = chrono::duration\_cast<chrono::nanoseconds>(stop0start).count();

Total +=diff;

If(I == 0){

T1 =diff

Use OMP

Define how many threads

Have to give compiler flags

Ways to chose pivot:

1. Median of 3 (first, middle, last OR random 3)
2. Median of 5
3. First
4. Last
5. Middle
6. Median