Lab 4 Sorting Pt. 2

Michael Morikawa

April 2, 2020

Lab Questions

Question 1 What are some options for picking a pivot for quick sort? Which one do you recommend and why?
Some options for picking a pivot is choosing the last element to be the pivot, picking the middle element as the pivot, picking a random pivot, and using the median of the first, middle, and last elements as the pivot. I would choose the median of 3 since you don't have to use a random number generator on each recursive call and it still performs well.

Question 2 Given the quick sort algorithm provided in the book, what kind of list would end up with a worst-case running time of $O(n^2)$?
A sorted list would give the worst case runtime of $O(n^2)$

Source Code

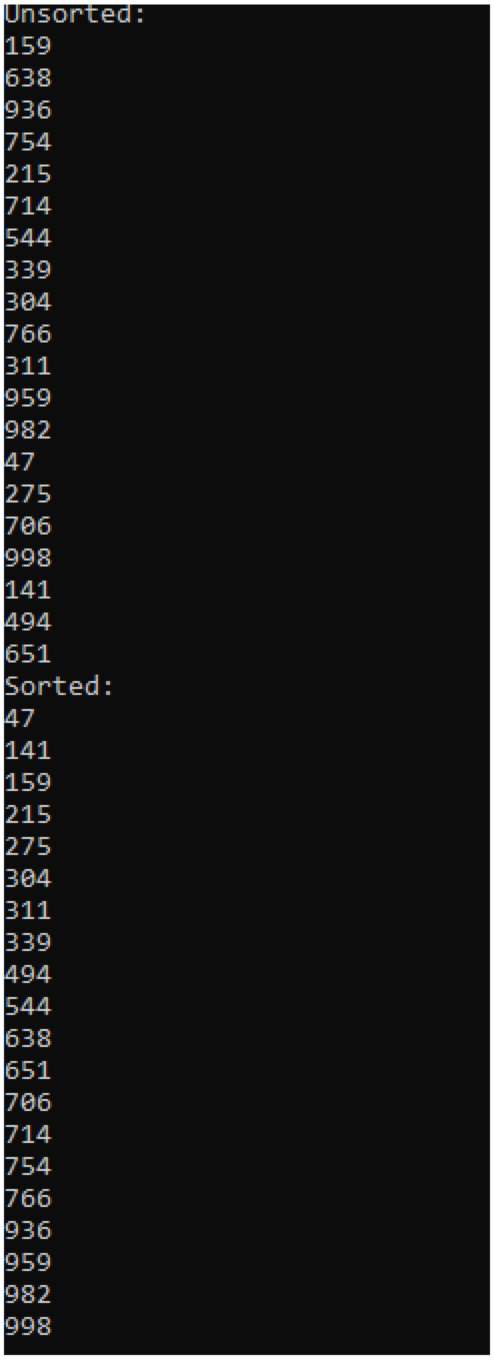
main.cpp

```
#include <iostream>
#include <vector>
#include <list>
#include "Comparators.hpp"

template <typename E, typename C>
void quickSort(std::vector<E> &S, const C &less);

template <typename E, typename C>
void quickSortStep(std::vector<E> &S, int a, int b, const C &less);

void printVector(const std::vector<int> &S);

void bucketSort(std::vector<int> &S)
{
    std::vector<std::list<int>> bucketArray(1000);
    int k;
    while (!S.empty())
    {
        k = S.back();
        S.pop_back();
        bucketArray[k].insert(bucketArray[k].end(), k);
    }
    for (int i = 0; i < 1000; i++)
    {
        while (!bucketArray[i].empty())
        {
```

```cpp
            S.push_back(bucketArray[i].front());
            bucketArray[i].pop_front();
        }
    }
}

std::vector<int> randomizeVector(int size)
{
    std::srand(time(NULL));
    std::vector<int> randVec;
    for (int i = 0; i < size; i++)
    {
        randVec.push_back(std::rand() % 1000);
    }

    return randVec;
}

int main()
{
    std::vector<int> quickSortTest = randomizeVector(20);
    std::vector<int> bucketSortTest = randomizeVector(100);
    std::cout << "Unsorted:\n";
    LowToHigh<int> less;
    printVector(quickSortTest);
    quickSort(quickSortTest, less);
    std::cout << "Sorted: \n";
    printVector(quickSortTest);
    std::cout << "\n\n\nBucketSort:\n";
    std::cout << "Unsorted:\n";
    printVector(bucketSortTest);
    bucketSort(bucketSortTest);
    std::cout << "Sorted:\n";
    printVector(bucketSortTest);
}

template <typename E, typename C>
void quickSort(std::vector<E> &S, const C &less)
{
    if (S.size() <= 1)
    {
        return;
    }
    quickSortStep(S, 0, S.size() - 1, less);
}
template <typename E, typename C>
void quickSortStep(std::vector<E> &S, int a, int b, const C &less)
{
    if (a >= b)
        return;     // 0 or 1 left? done
    E pivot = S[b]; // select last as pivot
    int l = a;      // left edge
    int r = b - 1;  // right edge
    while (l <= r)
    {
        while (l <= r && !less(pivot, S[l]))
            l++; // scan right till larger
        while (r >= l && !less(S[r], pivot))
            r--;   // scan left till smaller
        if (l < r) // both elements found
            std::swap(S[l], S[r]);
    }                                  // until indices cross
    std::swap(S[l], S[b]);             // store pivot at l
    quickSortStep(S, a, l - 1, less); // recur on both sides
    quickSortStep(S, l + 1, b, less);
}

void printVector(const std::vector<int> &S)
{
    for (int i : S)
    {
        std::cout << i << '\n';
    }
}
```

Output

QuickSort



```
Unsorted:
159
638
936
754
215
714
544
339
304
766
311
959
982
47
275
706
998
141
494
651
Sorted:
47
141
159
215
275
304
311
339
494
544
638
651
706
714
754
766
936
959
982
998
```

Bucket Sort

Note: Did file redirection to get output for bucket sort since its too large for screen shot
Unsorted: 823
368
269
387

308
653
283
420
310
780
678
126
134
412
239
366
349
106
356
380
688
595
338
542
488
230
223
800
86
350
782
909
718
404
649
26
57
932
799
719
64
477
846
198
241
437
917
590
543
273
323
583
221
661
125
709
243
700

```
861
330
51
996
239
121
400
240
500
809
173
651
881
589
480
79
788
722
516
705
664
411
330
339
346
551
353
824
260
596
524
474
926
575
470
518
49
870
758
549
31
283
Sorted:
26
31
49
51
57
64
79
86
106
121
125
```

126
134
173
198
221
223
230
239
239
240
241
243
260
269
273
283
283
308
310
323
330
330
338
339
346
349
350
353
356
366
368
380
387
400
404
411
412
420
437
470
474
477
480
488
500
516
518
524
542
543
549
551
575
583

589
590
595
596
649
651
653
661
664
678
688
700
705
709
718
719
722
758
780
782
788
799
800
809
823
824
846
861
870
881
909
917
926
932
996