Answer 1

Quicksort in CPP

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <list>
#include <set>
#include <chrono>
#include <numeric>
using namespace std;
using namespace std::chrono;

void print_vector_1d(vector<int> A){
  // helper function
  for (auto i : A){
    cout<<i<<" ";
  }
  cout<<endl;
}

int _partition_(vector<int> &A, int l, int r){
  // find random pivot
  srand(time(NULL));
  int random_ind = l + rand() % (r - l);
  // swap it with first  element
  swap(A[random_ind],A[l]);
  int p=A[l];
  int i=l;
  for (int j=l+1;j<r+1;j++){
    if (A[j]<=p){
      i=i+1;
      swap(A[i],A[j]);
    }
  }
  swap(A[i],A[l]);
  // return the pivot position
  return i;
}

void quicksort(vector<int> &A, int l, int r){
  if (l<r){
    int p=_partition_(A,l,r);
    // pivot p is in its rightful place in the vector A. Now recurse on the remaining parts of A
    quicksort(A,l,p-1);
    quicksort(A,p+1,r);
  }
}
```

```
int main(int argc, char const *argv[]) {
  vector<int> v(100) ;
  iota (begin(v), end(v), 1); // fill in 1-100 numbers in vector v
  for (int i=0;i<5;i++){
    auto start = high_resolution_clock::now();
    quicksort(v,0,v.size()-1);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "time for quicksort is "<<duration.count() << " ns"<<endl;

  }

  return 0;
}
```

Output-

> time for quicksort is 23601 ns
> time for quicksort is 13826 ns
> time for quicksort is 13791 ns
> time for quicksort is 13737 ns
> time for quicksort is 13666 ns

Answer 2

heap sort in CPP

```
#include <random>
#include <algorithm>
#include <iterator>
#include <iostream>
using namespace std;
void print_vector_1d(vector<int> A){
  // helper function
  for (auto i : A){
    cout<<i<<" ";
  }
  cout<<endl;
}


void max_heapify(vector<int> &A,int i,int heap_size){
  // fix one violation of heap property at node i
  int left_idx=2*i+1;
  int right_idx=2*i+2;
  int largest;
  if (left_idx<=heap_size and A[left_idx]>A[i]){
    largest=left_idx;
  }else{
    largest=i;
  }
  if (right_idx<=heap_size and A[right_idx]>A[largest]){
    largest=right_idx;
  }
  if (largest!=i){
    swap(A[i],A[largest]);
    max_heapify(A,largest,heap_size);
  }
  //print_vector_1d (A);
}

void build_max_heap(vector<int> &A,int heap_size){
  // Initialize heap. Runs in O(n) time
  for (int i=int(heap_size/2)-1;i>-1;--i){
    max_heapify(A,i,heap_size);
  }
}

void heap_sort(vector<int> &A){
  // first create the heap via build_max_heap
  int heap_size=A.size()-1;
  build_max_heap(A,A.size()-1);
  // extract max one at a time
```

```
  for (int i=A.size()-1;i>-1;--i){
    swap(A[0],A[i]);
    heap_size=heap_size-1;
    max_heapify(A,0,heap_size);
  }
}

int main()
{
  // Create a vector with 1-100 numbers
    vector<int> A;
                for (int i=1; i<101; ++i){
           A.push_back(i);
         }
    random_device rd;
    mt19937 g(rd());
  // Shuffle the vector with 1-100 numbers
    shuffle(A.begin(), A.end(), g);
                cout <<"Shuffled permutation is ";
                print_vector_1d (A);

  // Call heap_sort
    heap_sort(A);
    cout <<"Sorted permutation is ";
    print_vector_1d (A);
}
```

Output

> Shuffled permutation is
> 56 69 27 11 87 17 50 45 89 62 98 5 24 13 96 66 38 7 3 73 70 59 90 79 71 54 23 57 61 81 72 91 67 77 68 36 48 88 64 9 10 20 1 29 12 4 39 93 49 58 92 41 44 33 37 55 75 40 63 21 2 83 22 6 18 30 52 42 47 25 65 80 100 97 32 74 16 99 19 60 8 28 53 84 95 85 82 51 31 15 34 35 86 78 94 43 14 26 46 76

> Sorted permutation is
> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Answer 3

counting_sort code in C++

```
#include <vector>
#include <iostream>

using namespace std;

void print_vector_1d(vector<int> A){
  //helper function
  cout <<"Sorted output is ";
  for (auto i : A){
    cout<<i<<" ";
  }
  cout<<endl;
}

vector<int> counting_sort(vector<int> A){
  vector<int> C(*max_element(A.begin(), A.end())+1,0); //initialize C
  vector<int> B(A.size(),0); //initialize B. B has output elements
  for (auto i=0;i<A.size();++i){
    C[A[i]]=C[A[i]]+1;
  }

  for (auto i=1;i<C.size();++i){
    C[i]=C[i]+C[i-1];  // cummulative sum
  }m

  for (int i=A.size()-1;i>-1;i=i-1){
    B[C[A[i]]-1]=A[i];
    C[A[i]]=C[A[i]]-1;
  }
  return B; // final array
}

int main(int argc, char const *argv[]) {
  vector<int> A{20, 18, 5, 7, 16, 10, 9, 3, 12, 14, 0};
  vector<int> B=counting_sort(A);
  print_vector_1d(B);
  return 0;
}
```

> Output - 0 3 5 7 9 10 12 14 16 18 20

Answer 4

radix_sort code in C++

```
#include <vector>
#include <iostream>
#include <math.h>
using namespace std;
void print_vector_1d(vector<int> A){
  cout <<"Sorted answer is ";
  for (auto i : A){
    cout<<i<<" ";
  }
  cout<<endl;
}

vector<int> counting_sort(vector<int> A,int j){
  /*
  A has array of numbers. j represents the digit of element of A on which count sort will operate
  For eg.
  A=[329, 457, 657, 839, 436, 720, 353]; j=0. Count sort will work on LSB
  A=[329, 457, 657, 839, 436, 720, 353]; j=1. Count sort will work on middle bit(digit)
  A=[329, 457, 657, 839, 436, 720, 353]; j=2. Count sort will work on MSB
  */
  vector<int> C(10,0); //initialize C. 0-9 digits
  vector<int> B(A.size(),0); //#initialize B. B is output array
  for (auto i=0;i<A.size();++i){
    int _num_=int(A[i]/pow(10,j));
    _num_=_num_%10; // extract jth digit from element of A
    C[_num_]=C[_num_]+1; // count the number of times this jth digit comes
  }

  for (auto i=1;i<C.size();++i){
    C[i]=C[i]+C[i-1];
  }

  for (int i=A.size()-1;i>-1;i=i-1){

    int _num_=int(A[i]/pow(10,j));
    _num_=_num_%10; // extract jth digit from element of A
    B[C[_num_]-1]=A[i]; // arrange elements in output array based on C indexed by jth bit of element of A
    C[_num_]=C[_num_]-1;
  }
  //print_vector_1d (B); // uncomment to see counting sort is stable sort ! (And works correctly)
  return B;
}

vector<int> radix_sort(vector<int> A){
  // Figure how many digits are there in the elements of A. Given by max_digits.
  int max_digits=0;
```

```
  for (int i:A){
    int digits=0;
    while(i!=0){
      i=(int) i/10;
      digits=digits+1;
    }
    if (digits>max_digits){
      max_digits=digits;
    }
  }
  // call counting_sort for every digit of element of A, starting with LSB.
  // corresponds to j=0 below
  for (auto j=0;j<max_digits;++j){
    A=counting_sort(A,j);
  }
  return A;
}

int main(int argc, char const *argv[]) {
  vector<int> A{329, 457, 657, 839, 436, 720, 353};
  vector<int> B=radix_sort(A);
  print_vector_1d(B);
  return 0;
}
```

> Output- 329 353 436 457 657 720 839