Practical:- 01

Aim:-

WAP to implement linear search and determine its time complexity.

Code:-

```
#include <bits/stdc++.h>

using namespace std;

int main()
{
    int n, x, start, range, random;

    cout << "Enter the size of Array : ";
    cin >> n;
    vector<int> a(n);

    srand((unsigned)time(NULL));
    cout << "Starting Point : ";
    cin >> start;

    cout << "Range : ";
    cin >> range;

    for (int i = 0; i < n; i++)
    {
        random = rand() % range + start;
        a[i] = random;
        cout << random << " ";
    }
    cout << endl <<  "Enter the value to search : ";
    cin >> x;

    clock_t clock_start = clock();

    // linear search
    int i, flag = 0;
    for (i = 0; i < n; i++)
    {
        if (a[i] == x)
        {
            flag = 1;
            break;
        }
    }
    if (flag = 1)
```

```
        cout << "Found at Position: " << i+1 << endl;
    else
        cout << "Not found"<<endl;

    clock_t clock_end = clock();

    cout << "Execution time :" << fixed << (float)(clock_end - clock_start)/CLOCKS_PER_SEC <<
endl;
    return 0;
}
```

Output:-

```
Enter the size of Array : 200
Starting Point : 1
Range : 200
144 54 124 3 60 102 144 79 63 157 184 31 120 106 152 75 29 117 44 145 31 16 103 40 46 134 84 142 24
193 28 119 47 103 74 58 4 17 136 18 126 72 49 45 177 152 71 158 68 67 54 50 82 156 42 80 41 125 173
64 70 200 135 116 102 160 125 58 176 12 75 53 83 75 49 12 26 120 121 45 186 174 47 19 81 88 98 121 1
64 22 137 33 21 23 100 75 182 176 84 109 188 110 114 22 185 162 33 162 33 105 159 170 30 157 189 62
196 38 183 111 60 71 96 32 45 195 58 178 123 93 86 110 3 199 131 139 113 116 100 145 20 10 115 50 11
8 55 111 113 44 45 176 55 67 71 39 63 17 96 40 139 141 78 48 143 28 131 81 140 46 132 85 17 94 151 1
8 11 157 81 76 200 77 51 55 96 73 93 158 89 140 150 28 80 27 27 174 54 157 6 194 154
Enter the value to search : 194
Found at Position: 199
Execution time :0.000013
```

Practical:- 02

Aim:-

WAP to implement binary search and determine its time complexity.

Code:-

```
#include <bits/stdc++.h>

using namespace std;

int main()
{
    int n, x, start, range, random;

    cout << "Enter the size of Array : ";
    cin >> n;
    int a[n];

    srand((unsigned)time(NULL));
    cout << "Starting Point : ";
    cin >> start;

    cout << "Range : ";
    cin >> range;

    for (int i = 0; i < n; i++)
    {
        random = rand() % range + start;
        a[i] = random;
    }

    sort(a,a+n);

    for (int i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }

    int m, first, last, middle;
    cout<< endl << "Enter value to find: ";
    cin>>m;

    // binary search
    clock_t clock_start = clock();

    first = 0;
    last = n - 1;
```

```
    middle = (first + last) / 2;

    while (first <= last)
    {
        if (a[middle] < m)
        {
            first = middle + 1;
        }
        else if (a[middle] == m)
        {
            cout<<"Found at location: "<< middle+1<<endl;
            break;
        }
        else
        {
            last = middle - 1;
        }
        middle = (first + last) / 2;
    }
    if (first > last)
        cout<<"Not found"<<endl;

    clock_t clock_end = clock();

    cout << "Execution time :" << fixed << (float)(clock_end - clock_start)/CLOCKS_PER_SEC <<
endl;
    return 0;
}
```

Output:-

```
Enter the size of Array : 200
Starting Point : 1
Range : 200
2 2 3 3 4 5 6 6 6 7 9 11 12 14 14 14 15 15 16 17 17 17 17 19 20 21 21 22 23 27 30 30 32 32 33 35 35
37 37 38 39 40 40 44 44 45 45 48 50 51 51 51 52 54 54 55 56 56 58 58 60 62 63 63 63 64 65 68 69 70 7
0 71 71 74 74 75 76 76 76 77 77 78 78 79 79 79 80 80 81 81 82 82 83 87 87 89 89 89 89 89 90 90 90 91
 92 92 92 95 95 96 98 101 102 104 105 108 108 109 111 112 112 114 115 116 117 119 120 120 122 122 12
3 125 128 128 129 129 130 130 131 134 135 137 139 140 141 141 141 142 142 143 145 146 147 148 149 15
2 154 156 159 161 163 164 167 168 168 168 170 171 172 172 172 172 174 175 176 179 180 181 182 182 18
3 184 184 186 187 189 191 191 192 192 194 194 194 197 197 198 199 199 199 200
Enter value to find: 199
Found at location: 197
Execution time :0.000011
```

Practical:- 03

Aim:-

WAP to implement insertion sort and determine its time complexity.

Code:-

```
#include<bits/stdc++.h>

using namespace std;

void insertion_sort(int a[], int n)
{
    int i,j,temp;
    for(i=1;i<n;i++)
    {
        temp=a[i];
        j=i-1;

        while(j>=0 && a[j]>temp)
        {
            a[j+1]=a[j];
            j--;
        }
        a[j+1]=temp;
    }
}

int main()
{
    int n, start, range, random;

    cout << "Enter the size of Array : ";
    cin >> n;
    int a[n];

    srand((unsigned)time(NULL));
    cout << "Starting Point : ";
    cin >> start;

    cout << "Range : ";
    cin >> range;

    for (int i = 0; i < n; i++)
    {
        random = rand() % range + start;
        a[i] = random;
        cout << a[i] << " ";
```

```
    }

    clock_t clock_start = clock();
    insertion_sort(a,n);
    clock_t clock_end = clock();

    cout << endl << "Sorted array : " << endl;
    for(int i=0; i < n ; i++)
    {
        cout << a[i] << " ";
    }

    cout << endl << "Execution time : " << fixed << (double)(clock_end - clock_start) /
CLOCKS_PER_SEC << endl;
}
```

Output:-

```
Enter the size of Array : 200
Starting Point : 1
Range : 200
144 192 131 52 104 138 41 6 74 83 44 24 86 24 180 187 191 81 80 173 92 161 177 67 200 107 75 91 103
69 152 47 60 82 98 115 172 138 120 197 21 116 172 106 139 104 44 81 184 123 6 75 83 182 141 35 89 15
 125 191 83 77 37 94 110 86 160 81 24 79 78 196 146 1 53 36 104 97 117 87 171 122 161 6 103 53 40 19
1 19 116 134 53 192 122 146 102 8 105 134 183 136 163 130 33 164 134 69 67 182 185 154 153 106 66 11
0 160 119 101 103 89 16 36 142 160 109 39 13 116 144 146 50 31 109 131 63 24 65 83 90 198 67 195 102
 124 61 11 84 131 63 186 19 79 173 112 190 81 151 2 149 46 99 150 76 159 81 90 182 97 173 24 94 191
18 196 115 30 158 150 160 21 87 131 51 11 42 40 43 192 193 191 37 91 93 64 50 125 154 183 21 78
Sorted array :
1 2 6 6 6 8 11 11 13 15 16 18 19 19 21 21 21 24 24 24 24 24 30 31 33 35 36 36 37 37 39 40 40 41 42 4
3 44 44 46 47 50 50 51 52 53 53 53 60 61 63 63 64 65 66 67 67 67 69 69 74 75 75 76 77 78 78 79 79 80
 81 81 81 81 81 82 83 83 83 83 84 86 86 87 87 89 89 90 90 91 91 92 93 94 94 97 97 98 99 101 102 102
103 103 103 104 104 104 105 106 106 107 109 109 110 110 112 115 115 116 116 116 117 119 120 122 122
123 124 125 125 130 131 131 131 131 134 134 134 136 138 138 139 141 142 144 144 146 146 146 149 150
150 151 152 153 154 154 158 159 160 160 160 160 161 161 163 164 171 172 172 173 173 173 177 180 182
182 182 183 183 184 185 186 187 190 191 191 191 191 191 192 192 192 193 195 196 196 197 198 200
Execution time : 0.000031
```

Practical:- 04

Aim:-

WAP to implement bubble sort and determine its time complexity.

Code:-

```
#include<bits/stdc++.h>

using namespace std;

void sorting(int a[], int n){
    int i,j,temp;
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-1-i;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}

int main()
{
    int n, start, range, random;

    cout << "Enter the size of Array : ";
    cin >> n;
    int a[n];

    srand((unsigned)time(NULL));
    cout << "Starting Point : ";
    cin >> start;

    cout << "Range : ";
    cin >> range;

    for (int i = 0; i < n; i++)
    {
        random = rand() % range + start;
        a[i] = random;
        cout << a[i] << " ";
    }
```

```
    clock_t clock_start = clock();
    sorting(a, n);
    clock_t clock_end = clock();

    cout << endl << "Sorted array : " << endl;
    for(int i=0; i < n ; i++)
    {
        cout << a[i] << " ";
    }

    cout << endl << "Execution time : " << fixed << (double)(clock_end - clock_start) /
CLOCKS_PER_SEC << endl;
}
```

Output:-

```
Enter the size of Array : 200
Starting Point : 1
Range : 200
24 31 79 148 20 68 118 54 132 115 108 154 193 80 85 170 31 134 12 178 86 196 58 7 95 85 89 189 11 12
0 63 186 102 142 86 73 161 3 78 92 118 185 197 110 17 82 79 47 167 43 176 52 190 186 10 36 22 51 24
184 170 87 170 23 180 55 95 140 57 172 183 126 109 180 188 125 13 66 123 179 60 50 182 1 35 191 37 8
 193 60 192 114 98 161 136 77 167 182 168 23 106 103 101 14 82 88 138 94 105 12 24 165 61 5 117 96 1
95 153 55 188 165 46 101 62 158 189 91 124 170 10 99 75 112 199 40 145 86 177 38 142 188 61 106 49 1
7 175 96 12 127 150 151 43 148 3 57 105 143 99 181 113 108 79 139 172 29 179 116 66 107 106 7 47 118
 65 47 135 39 94 98 117 43 48 160 190 2 16 47 145 66 27 9 173 57 99 144 85 77 12 150 136 Sorted arra
y :
1 2 3 3 5 7 7 8 9 10 10 11 12 12 12 12 13 14 16 17 17 20 22 23 23 24 24 24 27 29 31 31 35 36 37 38 3
9 40 43 43 43 46 47 47 47 47 48 49 50 51 52 54 55 55 57 57 57 58 60 60 61 61 62 63 65 66 66 66 68 73
 75 77 77 78 79 79 79 80 82 82 85 85 85 86 86 86 87 88 89 91 92 94 94 95 95 96 96 98 98 99 99 99 101
 101 102 103 105 105 106 106 106 107 108 108 109 110 112 113 114 115 116 117 117 118 118 118 120 123
 124 125 126 127 132 134 135 136 136 138 139 140 142 142 143 144 145 145 148 148 150 150 151 153 154
 158 160 161 161 165 165 167 167 168 170 170 170 170 172 172 173 175 176 177 178 179 179 180 180 181
 182 182 183 184 185 186 186 188 188 188 189 189 190 190 191 192 193 193 195 196 197 199
Execution time : 0.000141
```

Practical:- 05

Aim:-

WAP to implement selection sort and determine its time complexity.

Code:-

```
#include<bits/stdc++.h>

using namespace std;

void selectionSort(int a[], int n){
    int i,j,k,y,x;

    for(i=0;i<n;i++)
    {
        x=a[i];
        for(j=i;j<n;j++)
        {
            if(x>a[j])
            {
                x=a[j];
                k=j;
            }
        }
        if(x!=a[i])
        {
            y=a[i];
            a[i]=x;
            a[k]=y;
        }
    }
}

int main()
{
    int n, start, range, random;

    cout << "Enter the size of Array : ";
    cin >> n;
    int a[n];

    srand((unsigned)time(NULL));
    cout << "Starting Point : ";
    cin >> start;

    cout << "Range : ";
    cin >> range;
```

```
    for (int i = 0; i < n; i++)
    {
        random = rand() % range + start;
        a[i] = random;
        cout << a[i] << " ";
    }

    clock_t clock_start = clock();
    selectionSort(a,n);
    clock_t clock_end = clock();

    cout << endl << "Sorted array : " << endl;
    for(int i=0; i < n ; i++)
    {
        cout << a[i] << " ";
    }

    cout << endl << "Execution time : " << fixed << (double)(clock_end -
clock_start)/CLOCKS_PER_SEC << endl;

    return 0;
}
```

Output:-

```
Enter the size of Array : 200
Starting Point : 1
Range : 200
119 177 102 177 158 86 37 19 138 26 99 27 90 21 138 106 114 138 114 67 157 75 152 17 114 152 26 84 1
79 155 9 49 131 63 26 40 148 14 10 37 40 60 63 81 80 200 186 193 138 52 59 46 126 162 14 191 114 40
26 44 194 35 45 76 97 70 67 44 35 76 80 74 135 142 107 167 93 92 159 182 143 170 27 20 131 193 10 19
6 184 188 192 177 22 36 4 70 57 22 65 91 97 96 117 183 189 23 149 81 66 60 63 9 29 41 180 111 33 142
 59 16 129 50 144 150 37 99 171 93 120 35 135 16 82 51 151 70 25 99 150 91 158 164 51 138 5 30 49 37
 171 59 5 99 108 100 200 144 199 122 188 118 156 122 86 37 125 36 58 149 86 8 39 44 171 41 181 175 7
1 181 164 41 39 120 92 98 19 91 193 17 13 180 87 120 54 172 109 178 159 166 78 44
Sorted array :
4 5 5 8 9 9 10 10 13 14 14 16 16 17 17 19 19 20 21 22 22 23 25 26 26 26 26 27 27 29 30 33 35 35 35 3
6 36 37 37 37 37 37 39 39 40 40 40 41 41 41 44 44 44 44 45 46 49 49 50 51 51 52 54 57 58 59 59 59 60
 60 63 63 63 65 66 67 67 70 70 70 71 74 75 76 76 78 80 80 81 81 82 84 86 86 86 87 90 91 91 91 92 92
93 93 96 97 97 98 99 99 99 99 100 102 106 107 108 109 111 114 114 114 114 117 118 119 120 120 120 12
2 122 125 126 129 131 131 135 135 138 138 138 138 138 142 142 143 144 144 148 149 149 150 150 151 15
2 152 155 156 157 158 158 159 159 162 164 164 166 167 170 171 171 171 172 175 177 177 177 178 179 18
0 180 181 181 182 183 184 186 188 188 189 191 192 193 193 193 194 196 199 200 200
Execution time : 0.000062
```

Practical:- 06

Aim:-

WAP to implement quick sort and determine its time complexity.

Code:-

```
#include<bits/stdc++.h>

using namespace std;

int partition(int a[],int start,int end)
{
    int pivot =a[start];
    int i=start;
    int j=end;
    while(i<j)
    {
        while(a[i]<=pivot)
        i++;
        while(a[j]>pivot)
        j--;
        if(i<j)
        {
        int temp=a[i];
        a[i]=a[j];
        a[j]=temp;
        }
    }
    int temp=a[j];
    a[j]=a[start];
    a[start]=temp;
    return j;
}

void quickSort(int a[],int start,int end)
{
    if(start<end)
    {
        int p=partition(a,start,end);
        quickSort(a,start,p-1);
        quickSort(a,p+1,end);
    }
}

int main()
{
    int n, start, range, random;
```

```
    cout << "Enter the size of Array : ";
    cin >> n;
    int a[n];

    srand((unsigned)time(NULL));
    cout << "Starting Point : ";
    cin >> start;

    cout << "Range : ";
    cin >> range;

    for (int i = 0; i < n; i++)
    {
        random = rand() % range + start;
        a[i] = random;
        cout << a[i] << " ";
    }

    clock_t clock_start = clock();
    quickSort(a, 0, n-1);
    clock_t clock_end = clock();

    cout << endl << "Sorted array : " << endl;
    for(int i=0; i < n ; i++)
    {
        cout << a[i] << " ";
    }

    cout << endl << "Execution time : " << (double)(clock_end - clock_start)/CLOCKS_PER_SEC <<
endl;
}
```

Output:-

```
Enter the size of Array : 200
Starting Point : 1
Range : 200
162 66 121 158 89 21 113 144 197 68 139 82 93 145 132 104 183 193 109 74 46 158 78 161 68 10 72 170
174 147 166 87 164 38 45 4 59 109 148 7 176 38 41 20 183 172 124 117 117 32 143 162 141 20 74 161 18
2 98 82 155 44 47 41 8 85 37 163 95 145 62 53 73 100 93 92 34 65 15 102 181 198 44 94 91 16 119 3 19
7 16 84 103 12 82 95 171 118 84 133 12 28 195 17 100 46 109 192 31 125 158 132 105 108 128 150 150 1
43 21 104 91 188 187 193 199 20 39 121 138 122 6 101 102 152 117 1 149 26 144 131 150 54 62 7 113 18
9 108 62 83 128 165 173 116 103 165 66 122 156 139 59 77 96 112 178 47 28 131 147 5 26 77 155 79 90
113 191 79 172 4 113 100 168 86 167 70 2 184 144 157 122 2 186 17 65 115 15 93 197 113 97 23 189
Sorted array :
1 2 2 3 4 4 5 6 7 7 8 10 12 12 15 15 16 16 17 17 20 20 20 21 21 23 26 26 28 28 31 32 34 37 38 38 39
41 41 44 44 45 46 46 47 47 53 54 59 59 62 62 62 65 65 66 66 68 68 70 72 73 74 74 77 77 78 79 79 82 8
2 82 83 84 84 85 86 87 89 90 91 91 92 93 93 93 94 95 95 96 97 98 100 100 100 101 102 102 103 103 104
 104 105 108 108 109 109 109 112 113 113 113 113 113 115 116 117 117 117 118 119 121 121 122 122 122
 124 125 128 128 131 131 132 132 133 138 139 139 141 143 143 144 144 144 145 145 147 147 148 149 150
 150 150 152 155 155 156 157 158 158 158 161 161 162 162 163 164 165 165 166 167 168 170 171 172 172
 173 174 176 178 181 182 183 183 184 186 187 188 189 189 191 192 193 193 195 197 197 197 198 199
Execution time : 0.000020
```

Practical:- 07

Aim:-

WAP to implement merge sort and determine its time complexity.

Code:-

```
#include <bits/stdc++.h>

using namespace std;
void merge(int a[],int low ,int mid,int high)
{
    int b[1000];
    int i=low;
    int j=mid+1;
    int k=low;
    while(i<=mid && j<=high)
    {
        if(a[i]<a[j])
        {
        b[k]=a[i];
        i++;
        }
        else{
            b[k]=a[j];
            j++;
        }
        k++;
    }
    if(i>mid){
        while(j<=high)
        {
            b[k]=a[j];
            j++;
            k++;
        }
    }
    else{
        while(i<=mid)
        {
            b[k]=a[i];
            i++;
            k++;
        }
    }
    for(k=low;k<=high;k++)
    {
        a[k]=b[k];
```

```
    }
}
void merge_sort(int a[],int low,int high)
{
    if(low<high)
    {
        int mid=(low+high)/2;
        merge_sort(a,low,mid);
        merge_sort(a,mid+1,high);
        merge(a,low,mid,high);
    }
}

int main()
{
    int n, start, range, random;

    cout << "Enter the size of Array : ";
    cin >> n;
    int a[n];

    srand((unsigned)time(NULL));
    cout << "Starting Point : ";
    cin >> start;

    cout << "Range : ";
    cin >> range;

    for (int i = 0; i < n; i++)
    {
        random = rand() % range + start;
        a[i] = random;
        cout << a[i] << " ";
    }

    clock_t clock_start = clock();
    merge_sort(a, 0, n-1);
    clock_t clock_end = clock();

    cout << endl << "Sorted array : " << endl;
    for(int i=0; i < n ; i++)
    {
        cout << a[i] << " ";
    }

    cout << endl << "Execution time : " << (double)(clock_end - clock_start)/CLOCKS_PER_SEC <<
endl;
    return 0;
}
```

Output:-

```
Enter the size of Array : 200
Starting Point : 1
Range : 200
19 48 196 160 148 127 6 147 55 67 28 31 188 47 25 63 117 49 139 145 173 87 136 11 101 69 194 138 100
 131 56 70 130 51 29 30 177 186 176 31 53 4 13 40 50 37 54 166 38 192 63 162 30 198 172 131 18 117 2
0 70 47 75 139 176 125 168 5 101 153 133 84 157 136 96 148 137 85 2 54 122 145 116 35 175 65 6 57 83
 122 76 152 120 150 90 95 26 9 52 78 114 184 161 70 71 9 18 7 93 171 12 166 115 128 200 41 144 5 97
178 78 172 129 197 73 171 43 50 179 46 128 44 181 40 114 51 48 83 9 92 53 21 57 119 100 8 160 43 164
 56 21 41 180 101 189 4 23 184 54 2 29 133 197 10 172 110 12 172 144 173 63 148 193 72 67 44 79 178
38 43 33 10 35 164 63 24 168 85 7 173 38 187 105 35 148 28 144 160 199 40 84
Sorted array :
2 2 4 4 5 5 6 6 7 7 8 9 9 9 10 10 11 12 12 13 18 18 19 20 21 21 23 24 25 26 28 28 29 29 30 30 31 31
33 35 35 35 37 38 38 38 40 40 40 41 41 43 43 43 44 44 46 47 47 48 48 49 50 50 51 51 52 53 53 54 54 5
4 55 56 56 57 57 63 63 63 63 65 67 67 69 70 70 70 71 72 73 75 76 78 78 79 83 83 84 84 85 85 87 90 92
 93 95 96 97 100 100 101 101 101 105 110 114 114 115 116 117 117 119 120 122 122 125 127 128 128 129
 130 131 131 133 133 136 136 137 138 139 139 144 144 144 145 145 147 148 148 148 148 150 152 153 157
 160 160 160 161 162 164 164 166 166 168 168 171 171 172 172 172 172 173 173 173 175 176 176 177 178
 178 179 180 181 184 184 186 187 188 189 192 193 194 196 197 197 198 199 200
Execution time : 0.000022
```

Practical:- 08

Aim:-

WAP to implement heap sort and determine its time complexity.

Code:-

```
#include <bits/stdc++.h>

using namespace std;

void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

int main()
{
    int n, start, range;

    cout << "Enter the size of Array : ";
    cin >> n;
    int a[n];

    srand((unsigned)time(NULL));
    cout << "Starting Point : ";
    cin >> start;

    cout << "Range : ";
```

```
    cin >> range;

    for (int i = 0; i < n; i++)
    {
        int random = rand() % range + start;
        a[i] = random;
        cout << a[i] << " ";
    }

    clock_t clock_start = clock();
    heapSort(a, n);
    clock_t clock_end = clock();

    cout << endl << "Sorted array : " << endl;
    for(int i=0; i < n ; i++)
    {
        cout << a[i] << " ";
    }

    cout << endl << "Execution time : " << fixed << (double)(clock_end -
clock_start)/CLOCKS_PER_SEC << endl;
    return 0;
}
```

Output:-

```
Enter the size of Array : 200
Starting Point : 1
Range : 200
107 122 79 184 134 104 98 155 88 123 34 200 181 101 119 8 164 105 132 53 51 136 118 16 159 118 134 92
177 198 136 83 119 167 18 4 70 68 110 110 190 143 109 123 196 179 82 159 83 13 163 134 149 32 149 59 1
01 35 151 77 32 38 111 102 4 129 57 26 196 166 135 137 109 195 59 56 173 141 14 7 153 128 140 53 159 4
1 64 60 27 166 136 58 3 199 111 7 79 167 32 26 132 118 162 192 112 173 47 36 113 12 42 17 139 134 22 5
0 126 85 109 152 2 196 161 4 194 23 162 24 189 145 49 72 14 11 64 77 135 62 112 47 26 106 15 164 191 1
88 165 116 24 73 19 25 69 179 181 14 153 94 38 141 191 86 12 4 48 27 81 182 89 144 180 66 1 147 181 19
1 134 146 106 110 18 124 134 38 54 66 52 6 160 41 98 150 78 62 153 126 88 185 59 128
Sorted array :
1 2 3 4 4 4 4 6 7 7 8 11 12 12 13 14 14 14 15 16 17 18 18 19 22 23 24 24 25 26 26 26 27 27 32 32 32 34
 35 36 38 38 38 41 41 42 47 47 48 49 50 51 52 53 53 54 56 57 58 59 59 59 60 62 62 64 64 66 66 68 69 70
 72 73 77 77 78 79 79 81 82 83 83 85 86 88 88 89 92 94 98 98 101 101 102 104 105 106 106 107 109 109 1
09 110 110 110 111 111 112 112 113 116 118 118 118 119 119 122 123 123 124 126 126 128 128 129 132 132
 134 134 134 134 134 134 135 135 136 136 136 137 139 140 141 141 143 144 145 146 147 149 149 150 151 1
52 153 153 153 155 159 159 159 160 161 162 162 163 164 164 165 166 166 167 167 173 173 177 179 179 180
 181 181 181 182 184 185 188 189 190 191 191 191 192 194 195 196 196 196 198 199 200
Execution time : 0.000043
```

Practical:- 09

Aim:-

Compare time complexities of all sorting algorithm.

Code:-

```
#include <iostream>

using namespace std;

#define size 2000

struct Array{
    int arr[size];
};

void heapify(int a[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && a[l] > a[largest])
        largest = l;
    if (r < n && a[r] > a[largest])
        largest = r;
    if (largest != i) {
        swap(a[i], a[largest]);
        heapify(a, n, largest);
    }
}
void heapSort(struct Array temp, int n)
{
    int *a = temp.arr;
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(a, n, i);
    for (int i = n - 1; i > 0; i--) {
        swap(a[0], a[i]);
        heapify(a, i, 0);
    }
}

void printArray(int arr[], int n)
{
    cout<<endl<<"Sorted Elements: ";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}
```

```
void merge(struct Array temp,int low ,int mid,int high)
{
    int *a = temp.arr;
    int b[1000];
    int i=low;
    int j=mid+1;
    int k=low;
    while(i<=mid && j<=high)
    {
        if(a[i]<a[j])
        {
        b[k]=a[i];
        i++;
        }
        else{
            b[k]=a[j];
            j++;
        }
        k++;
    }
    if(i>mid){
        while(j<=high)
        {
            b[k]=a[j];
            j++;
            k++;
        }
    }
    else{
        while(i<=mid)
        {
            b[k]=a[i];
            i++;
            k++;
        }
    }
    for(k=low;k<=high;k++)
    {
        a[k]=b[k];
    }
}
void merge_sort(struct Array a,int low,int high)
{
    if(low<high)
    {
        int mid=(low+high)/2;
        merge_sort(a,low,mid);
        merge_sort(a,mid+1,high);
        merge(a,low,mid,high);
```

```
    }
}

int partition(struct Array temp,int start,int end)
{
    int *a = temp.arr;
    int pivot =a[start];
    int i=start;
    int j=end;
    while(i<j)
    {
        while(a[i]<=pivot)
        i++;
        while(a[j]>pivot)
        j--;
        if(i<j)
        {
        int temp=a[i];
        a[i]=a[j];
        a[j]=temp;
        }
    }
    int tem=a[j];
    a[j]=a[start];
    a[start]=tem;
    return j;
}

void quick_sort(struct Array a,int start,int end)
{
    if(start<end)
    {
        int p=partition(a,start,end);
        quick_sort(a,start,p-1);
        quick_sort(a,p+1,end);
    }
}

void insertion_sort(struct Array temp,int n)
{
    int *a = temp.arr;
    int j,tem;
    for(int i=1;i<n;i++)
    {
        int tem=a[i];
         j=i-1;
        while(j>=0 && a[j]>tem)
        {
        a[j+1]=a[j];
        j--;
```

```
        }
    a[j+1]=tem;
    }
}

void bubble_sort(struct Array temp,int n)
{
    int *a = temp.arr;
    for(int i=0;i<n-1;i++)
    {
        for(int j=0;j<n-i-1;j++)
        {
            if(a[j+1]<a[j])
            {
                int temp;
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;

            }

        }
    }
}

void selectionSort(struct Array temp, int n){
    int *a = temp.arr;
    int i,j,k,y,x;

    for(i=0;i<n;i++)
    {
        x=a[i];
        for(j=i;j<n;j++)
        {
            if(x>a[j])
            {
                x=a[j];
                k=j;
            }
        }
        if(x!=a[i])
        {
            y=a[i];
            a[i]=x;
            a[k]=y;
        }
    }
}

int main()
```

```
{
    int n,offset,range;
    cout<<"Enter the size of Array :";
    cin>>n;
    srand((unsigned) time(NULL));

    struct Array obj;

    cout<<"\nEnter the starting value :";
    cin>>offset;
    cout<<"\nEnter the Range : ";
    cin>>range;
    for(int i=0;i<n;i++)
    {
        int random = offset + (rand()%range);
        obj.arr[i]=random;
    }
    cout<<endl;

clock_t clock_start,clock_end;
   clock_start=clock();
   heapSort(obj,n);
   clock_end=clock();
   float total_t=(float)(clock_end-clock_start)/CLOCKS_PER_SEC;

    clock_start=clock();
    merge_sort(obj,0,n-1);
    clock_end=clock();
    float total_t1=(float)(clock_end-clock_start)/CLOCKS_PER_SEC;

    clock_start=clock();
    quick_sort(obj,0,n-1);
    clock_end=clock();
    float total_t2=(float)(clock_end-clock_start)/CLOCKS_PER_SEC;

    clock_start=clock();
    insertion_sort(obj,n);
    clock_end=clock();
    float total_t3=(float)(clock_end-clock_start)/CLOCKS_PER_SEC;

     clock_start=clock();
     bubble_sort(obj,n);
     clock_end=clock();
    float total_t4=(float)(clock_end-clock_start)/CLOCKS_PER_SEC;

     clock_start = clock();
    selectionSort(obj,n);
     clock_end = clock();
    float total_t5=(float)(clock_end-clock_start)/CLOCKS_PER_SEC;
```

```
    cout<<"\nExecution Time of Heap Sort:"<<fixed<<total_t<<" sec";

    cout<<"\nExecution Time of Merge Sort:"<<fixed<<total_t1<<" sec";

    cout<<"\nExecution Time of Quick Sort:"<<fixed<<total_t2<<" sec";

    cout<<"\nExecution Time of Insertion Sort:"<<fixed<<total_t3<<" sec";

    cout<<"\nExecution Time of Selection Sort:"<<fixed<<total_t5<<" sec";

    cout<<"\nExecution Time of Bubble Sort:"<<fixed<<total_t4<<" sec";

    return 0;
}
```

Output:-

```
Enter the size of Array :500

Enter the starting value :1

Enter the Range : 500


Execution Time of Heap Sort:0.000129 sec
Execution Time of Merge Sort:0.000328 sec
Execution Time of Quick Sort:0.000724 sec
Execution Time of Insertion Sort:0.000174 sec
Execution Time of Selection Sort:0.000323 sec
Execution Time of Bubble Sort:0.000474 sec
```

Practical:- 10

Aim:-

WAP to implement strassen's matrix multiplication

Code:-

```
#include <algorithm>
#include <iostream>
#include <bits/stdc++.h>

using namespace std;

void display(vector< vector<int>> &matrix, int m, int n){
    for (int i = 0; i < m; i++){
        for (int j = 0; j < n; j++){
            if (j != 0){
                cout << "\t";
            }
            cout << matrix[i][j];
        }
        cout << endl;
    }
}

void multiplyarr(vector<vector<int>> &a, vector<vector<int>> &b, int m, int n)
{
    vector<int> temp(m);
    vector<vector<int>> c(n, temp);
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
          c[i][j]=0;
          for(int k=0;k<m;k++)
          {
              c[i][j]+=a[i][k]*b[k][j];
          }
        }
    }
    display(c,3,3);
}


int nextpowerof2(int k){
    return pow(2, int(ceil(log2(k))));
}
```

```
void add(vector<vector<int>> &A, vector<vector<int>> &B, vector<vector<int>> &C, int size){
    int i, j;
    for (i = 0; i < size; i++){
        for (j = 0; j < size; j++){
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}

void sub(vector<vector<int>> &A, vector<vector<int>> &B, vector<vector<int>> &C, int size){
    int i, j;
    for (i = 0; i < size; i++){
        for (j = 0; j < size; j++){
            C[i][j] = A[i][j] - B[i][j];
        }
    }
}

void Strassen_algorithmA(vector<vector<int>> &A, vector<vector<int>> &B, vector<vector<int>> &C,
int size)
{
    if (size == 1)
    {
        C[0][0] = A[0][0] * B[0][0];
        return;
    }
    else
    {
        int new_size = size / 2;
        vector<int> z(new_size);
        vector<vector<int>>
            a11(new_size, z), a12(new_size, z), a21(new_size, z), a22(new_size, z),
            b11(new_size, z), b12(new_size, z), b21(new_size, z), b22(new_size, z),
            c11(new_size, z), c12(new_size, z), c21(new_size, z), c22(new_size, z),
            p1(new_size, z), p2(new_size, z), p3(new_size, z), p4(new_size, z),
            p5(new_size, z), p6(new_size, z), p7(new_size, z),
            aResult(new_size, z), bResult(new_size, z);

int i, j;

for (i = 0; i < new_size; i++)
        {
            for (j = 0; j < new_size; j++)
            {
                a11[i][j] = A[i][j];
                a12[i][j] = A[i][j + new_size];
                a21[i][j] = A[i + new_size][j];
                a22[i][j] = A[i + new_size][j + new_size];

                b11[i][j] = B[i][j];
```

```
            b12[i][j] = B[i][j + new_size];
            b21[i][j] = B[i + new_size][j];
            b22[i][j] = B[i + new_size][j + new_size];
        }
}

        add(a11, a22, aResult, new_size);
        add(b11, b22, bResult, new_size);
        Strassen_algorithmA(aResult, bResult, p1, new_size);

        add(a21, a22, aResult, new_size);
        Strassen_algorithmA(aResult, b11, p2, new_size);

        sub(b12, b22, bResult, new_size);
        Strassen_algorithmA(a11, bResult, p3, new_size);

        sub(b21, b11, bResult, new_size);
        Strassen_algorithmA(a22, bResult, p4, new_size);

        add(a11, a12, aResult, new_size);
        Strassen_algorithmA(aResult, b22, p5, new_size);

        sub(a21, a11, aResult, new_size);
        add(b11, b12, bResult, new_size);
        Strassen_algorithmA(aResult, bResult, p6, new_size);

        sub(a12, a22, aResult, new_size);
        add(b21, b22, bResult, new_size);
        Strassen_algorithmA(aResult, bResult, p7, new_size);

        add(p3, p5, c12, new_size);
        add(p2, p4, c21, new_size);

        add(p1, p4, aResult, new_size);
        add(aResult, p7, bResult, new_size);
        sub(bResult, p5, c11, new_size);

        add(p1, p3, aResult, new_size);
        add(aResult, p6, bResult, new_size);
        sub(bResult, p2, c22, new_size);

        for (i = 0; i < new_size; i++)
        {
            for (j = 0; j < new_size; j++)
            {
                C[i][j] = c11[i][j];
                C[i][j + new_size] = c12[i][j];
                C[i + new_size][j] = c21[i][j];
                C[i + new_size][j + new_size] = c22[i][j];
            }
```

```
        }
    }
}
void Strassen_algorithm(vector<vector<int>> &A, vector<vector<int>> &B, int m, int n, int a, int
b)
{
    int k = max({m, n, a, b});

    int s = nextpowerof2(k);

    vector<int> z(s);
    vector<vector<int>> Aa(s, z), Bb(s, z), Cc(s, z);

    for (unsigned int i = 0; i < m; i++)
    {
        for (unsigned int j = 0; j < n; j++)
        {
            Aa[i][j] = A[i][j];
        }
    }
    for (unsigned int i = 0; i < a; i++)
    {
        for (unsigned int j = 0; j < b; j++)
        {
            Bb[i][j] = B[i][j];
        }
    }
    Strassen_algorithmA(Aa, Bb, Cc, s);
    vector<int> temp1(b);
    vector<vector<int>> C(m, temp1);
    for (unsigned int i = 0; i < m; i++)
    {
        for (unsigned int j = 0; j < b; j++)
        {
            C[i][j] = Cc[i][j];
        }
    }
    display(C, m, b);
}

int main() {
    vector<vector<int>> a = {{1,2,3},
       {1,2,1},
       {1,1,1}};
    vector<vector<int>> b = {{1,2,3},
       {1,2,1},
       {1,1,1}};

       clock_t clock_start,clock_end;
       clock_start = clock();
```

```
    multiplyarr(a,b,3,3);
    clock_end = clock();
    float total_t=(float)(clock_end-clock_start)/CLOCKS_PER_SEC;
       cout<<"\nclock time for multiplication is :"<<fixed<<total_t<<endl;

       clock_t clock_start1,clock_end1;
       clock_start1=clock();
       Strassen_algorithm(a, b, 3, 3, 3, 3);
       clock_end1=clock();
       float total_t1=(float)(clock_end1-clock_start1)/CLOCKS_PER_SEC;
       cout<<"\nclock time for Strassen multiplication is :"<<fixed<<total_t1;
    return 0;
}
```

Output:-

```
6       9       8
4       7       6
3       5       5

clock time for multiplication is :0.000034
6       9       8
4       7       6
3       5       5

clock time for strassen multiplication is :0.000095
```