

Project 1 documentation

I have created class for linked list and array. I initialize the variable with some constant values and variables. In Class I have created the private variables for clock and counting some creation, initialization time with total values

Linked list :

We can snip the linked list.

Insertion and deletion is easy because of head pointer.

Linked list is consider as dynamic data structure because of no fixed size.

Sequential access is not possible. Extra pointer is required for storing the data.

Pointer operation is more complex because of random memory access

```
void LL_execution()// Linked list function
{
    int i,j;
    nStart_bit = clock(); //begin the clock
    Project1 *pList = NULL; // BaSe_Pointer Creation

    for(i = 1; i <= N; i++)
    {
        Project1* nCurr = new Project1;
        nCurr->n_value = rand()%10000;
        nCurr->pnext = pList;
        pList=nCurr;
    }//linked list creation and initialization

    nStop_bit = clock(); //Clock freeze
    nC_time = (nStop_bit - nStart_bit); //Linked List creation and initialization time
    cout << "Linked List :- \n";
    cout << "Creation Time:- " << nC_time << " ";

    nStart_bit = clock();///begin the clock
    for(j=i=0; i < sizeof(n_Ind)/sizeof(n_Ind[0]) ; i++)
    {
        Project1* nCurr = new Project1;
        nCurr->n_value = rand()%10000; //storing random value between 0 and 10000
        nCurr->pnext = NULL;
        Project1 * pPrev=pList;
        //code for inserting as head
        if(n_Ind[i]==0)
        {
            nCurr->pnext=pList;
            pList=nCurr;
            continue;
        }
        j=1;
```

```

        //code traveling till the index position
        while(n_Ind[i]>1 && j< n_Ind[i])
        {
            pPrev=pPrev->pnext;
            j++;
        }
        nCurr->pnext=pPrev->pnext;
        pPrev->pnext=nCurr;

    }
    nStop_bit = clock(); //clock Freeze
    nIn_time = (nStop_bit-nStart_bit);    //Linked List insertion time

    cout<<"Insertion Time:- " << nIn_time <<" ";
    cout<<"\nLinked List (Total time = Creation + Insertion ):- " << nC_time + nIn_time <<"\n\n";
}

```

Array:

Array is static data structure and easy to create.

It has fixed size.

It can have sequential access and random access because of index position to the values

It holds similar type of data.

We can not change the values at runtime.

void A_execution()//Array function

```

{
    int i,j;
    nStart_bit = clock(); //clock start
    int *n_arr =new int [N + (sizeof(n_Ind)/sizeof(n_Ind[0]))]; //Array creation with extra size for
insertion
    for(i = 0 ; i < N ; i++)
    {
        n_arr[i] = rand()%10000;    //storing random value between 0 and 10000
    }
    nStop_bit = clock(); //clock stop
    nC_time = (nStop_bit-nStart_bit); //array creation and initialization time
    cout << "Array :- \n";
    cout<<"Creation time:- " << nC_time <<"    ";
    nStart_bit = clock(); //clock start
    for(i = 0 ; i < sizeof(n_Ind)/sizeof(n_Ind[0]) ; i++)//Insert array value at specific index position
    {
        for(j=N+i; j > n_Ind[i]; j--)
        {
            n_arr[j] = n_arr[j-1];
        }
        n_arr[n_Ind[i]]=rand()%10000;    //storing random value between 0 and 10000
    }

    nStop_bit = clock(); //clock stop

```

```

nIn_time = (nStop_bit-nStart_bit); //array insertion time
cout << "Insertion time:- " << nIn_time << "\n";
cout<<"Array (Total time = Creation + Insertion):- "<<nC_time + nIn_time<<"\n\n";
delete[] n_arr; //deleting array
}

```

Random number can be generated by using rand() function. In between given range

```

void generateRandomNumbers(int n_arr[],int N)//Random number generation function {
{
    srand((unsigned)time(0));//to generate random number

    for (int i = 0; i < N; i++)
    {
        n_arr[i] = (rand()%100) + 1;
    }
}

```

Results:-

Figure1. 20000 values for the comparison

The screenshot displays an IDE with a C++ project named 'Mayur_50155714_project1'. The code in 'Mayur_50155714_project1.cpp' defines a class 'project1' for comparing array and linked list performance. It sets a constant N = 20000 and uses 'rand()' for random number generation. The 'project1' class has private members for array creation time, insertion time, and linked list creation time, and public members for total time and insertion time. The output window shows the following results:

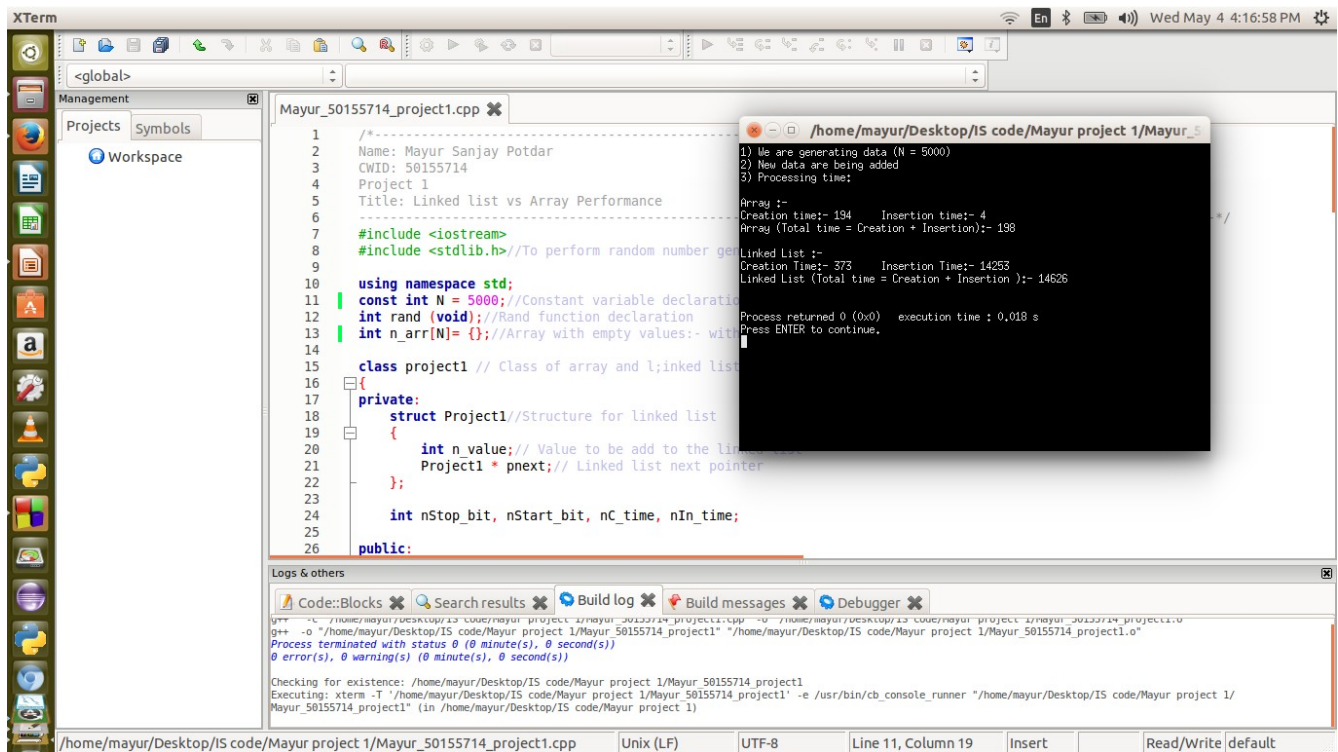
```

1) We are generating data (N = 20000)
2) New data are being added
3) Processing time:
Array :-
Creation time:- 774      Insertion time:- 95
Array (Total time = Creation + Insertion):- 869
Linked List :-
Creation Time:- 2416      Insertion Time:- 42105
Linked List (Total time = Creation + Insertion):- 44521
Process returned 0 (0x0)   execution time : 0.050 s
Press ENTER to continue.

```

The status bar at the bottom indicates the file path, encoding (UTF-8), and cursor position (Line 11, Column 17).

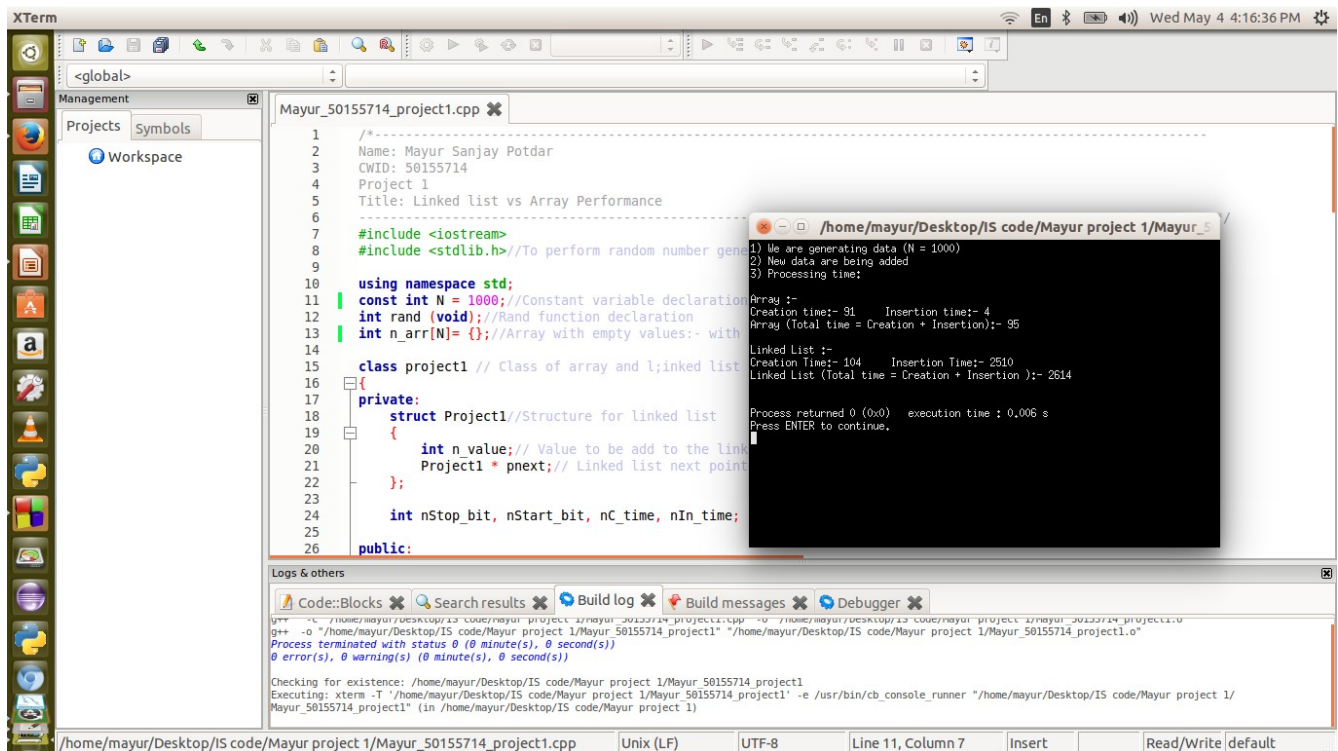
Figure 2. 5000 values for the comparison



The screenshot shows an XTerm window with a C++ project named 'Mayur_50155714_project1.cpp'. The code defines a 'project1' class for comparing array and linked list performance with N=5000. A terminal window displays the following output:

```
1) We are generating data (N = 5000)
2) New data are being added
3) Processing time:
Array :-
Creation time:- 194      Insertion time:- 4
Array (Total time = Creation + Insertion):- 198
Linked List :-
Creation Time:- 373      Insertion Time:- 14263
Linked List (Total time = Creation + Insertion):- 14626
Process returned 0 (0x0)   execution time : 0.018 s
Press ENTER to continue.
```

Figure3. 1000 values for the comparison



The screenshot shows the same XTerm window with the C++ project, but with N=1000. The terminal window displays the following output:

```
1) We are generating data (N = 1000)
2) New data are being added
3) Processing time:
Array :-
Creation time:- 91      Insertion time:- 4
Array (Total time = Creation + Insertion):- 95
Linked List :-
Creation Time:- 104      Insertion Time:- 2510
Linked List (Total time = Creation + Insertion):- 2614
Process returned 0 (0x0)   execution time : 0.006 s
Press ENTER to continue.
```

Figure 4. 100 values for the comparison

```

106     delete[] n_arr; //deletion of array
107
108 void generateRandomNumbers(int n_arr[],int N)//Random number generation function
109 {
110     srand((unsigned)time(0)); //to generate random number
111
112     for (int i = 0; i < N; i++)
113     {
114         n_arr[i] = (rand()%100) + 1;
115     }
116 }
117
118
119 int main()
120 {
121     project1 p;
122     cout<<"1) We are generating data (N = "<<N<<endl;
123     //call random number functions
124     p.generateRandomNumbers(n_arr,N);
125     cout<<"2) New data are being added\n";
126     cout<<"3) Processing time:\n\n";
127     p.A_execution(); //arrays using class ob
128     p.LL_execution(); //function linked list
129     return 0;
130 }
131

```

Mayur project 1

```

1) We are generating data (N = 100)
2) New data are being added
3) Processing time:
Array :-
Creation time:- 68      Insertion time:- 3
Array (Total time = Creation + Insertion):- 72
Linked List :-
Creation Time:- 13      Insertion Time:- 37
Linked List (Total time = Creation + Insertion ):- 50
Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.

```

Run: Debug in Mayur project 1 (compiler: GNU GCC Compiler)-----

Checking for existence: /home/mayur/Desktop/IS code/Mayur project 1/Mayur project 1

Executing: xterm -T Mayur\ project\ 1 -e /usr/bin/cb_console_runner LD_LIBRARY_PATH=LD_LIBRARY_PATH:.. /home/mayur/Desktop/IS code/Mayur\ project\ 1/bin/Debug/Mayur\ project\ 1 (in /home/mayur/Desktop/IS code/Mayur project 1/.)

Figure 4. 10000 values for the comparison

```

1  /*
2  Name: Mayur Sanjay Potdar
3  CWID: 50155714
4  Project 1
5  Title: Linked list vs Array Performance
6
7  #include <iostream>
8  #include <stdlib.h> //To perform random number generation
9
10 using namespace std;
11 const int N = 10000; //Constant variable declaration
12 int rand (void); //Rand function declaration
13 int n_arr[N] = {}; //Array with empty values:- with const
14
15 class project1 // Class of array and l;linked list
16 {
17 private:
18     struct Project1 //Structure for linked list
19     {
20         int n_value; // Value to be add to the linked li
21         Project1 * pnext; // Linked list next pointer
22     };
23     int nStop_bit, nStart_bit, nC_time, nIn_time;
24 public:

```

/home/mayur/Desktop/IS code/Mayur project 1/Mayur project 1

```

1) We are generating data (N = 10000)
2) New data are being added
3) Processing time:
Array :-
Creation time:- 324      Insertion time:- 47
Array (Total time = Creation + Insertion):- 371
Linked List :-
Creation Time:- 1019      Insertion Time:- 24545
Linked List (Total time = Creation + Insertion ):- 25564
Process returned 0 (0x0)   execution time : 0.029 s
Press ENTER to continue.

```

Process terminated with status 0 (0 minute(s), 0 second(s))

0 error(s), 0 warning(s) (0 minute(s), 0 second(s))

Checking for existence: /home/mayur/Desktop/IS code/Mayur project 1/Mayur 50155714.project1

Executing: xterm -T "/home/mayur/Desktop/IS code/Mayur project 1/Mayur 50155714.project1" -e /usr/bin/cb_console_runner "/home/mayur/Desktop/IS code/Mayur project 1/Mayur 50155714.project1" (in /home/mayur/Desktop/IS code/Mayur project 1/)

Performance analysis : - Array :-

Values	Time			Result
	Creation	Insertion	Total Time	
100	69	03	72	Worst Case
1000	91	04	95	Best Case
5000	191	04	198	Best Case
10000	324	47	371	Best Case
20000	774	95	869	Best Case

Linked List :-

Values	Time			Result
	Creation	Insertion	Total Time	
100	13	37	50	Best Case
1000	104	2150	2614	Worst Case
5000	373	14253	14626	Worst Case
10000	1019	24545	25564	Worst Case
20000	2416	42105	44521	Worst Case

Note : - For every execution values are changing according to the system clock

Flow chart difference and comparission

Figure 5. Creation with Initilization of array and linked list

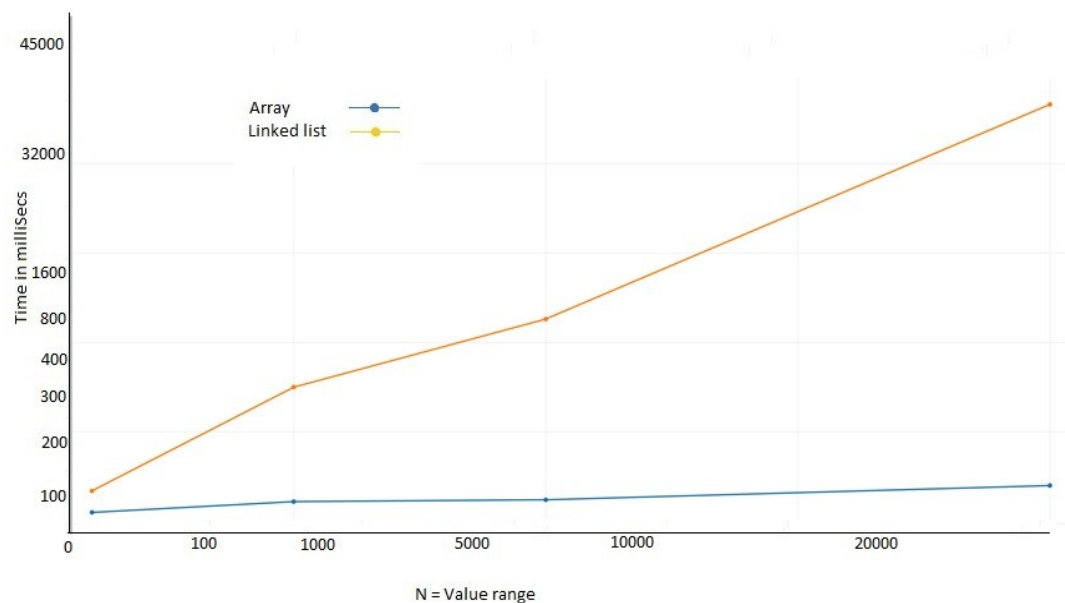


Figure 6. Value Insertion of array and linked list

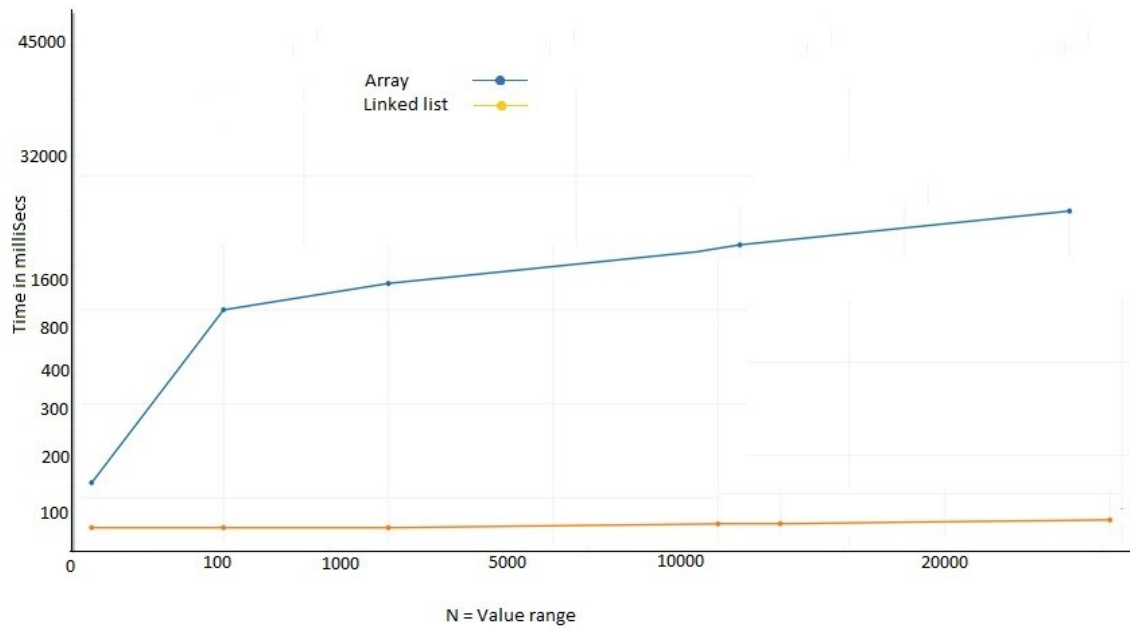
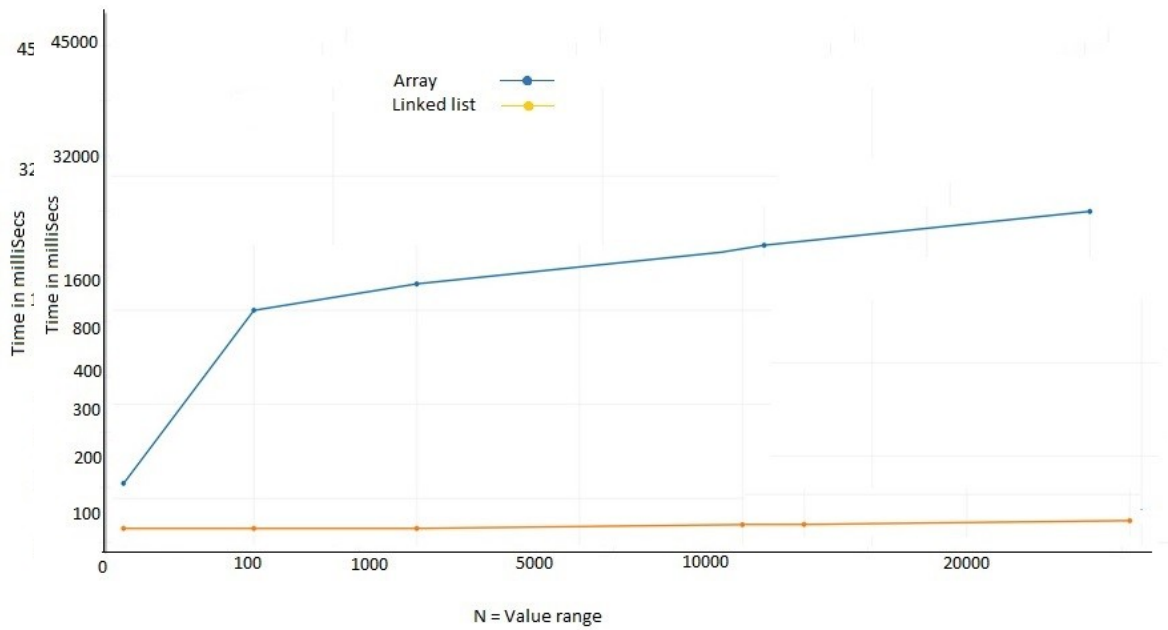


Figure 7. Final Result of array and linked list



Conclusion:

Creation and initialization of an array is pretty best as compare to Linked list. Where as Insertion and deletion of linked list element is easy than an array. If values are maximum then performance of linked list is worst. But if you know the size of an array then array performs really well than linked list.