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

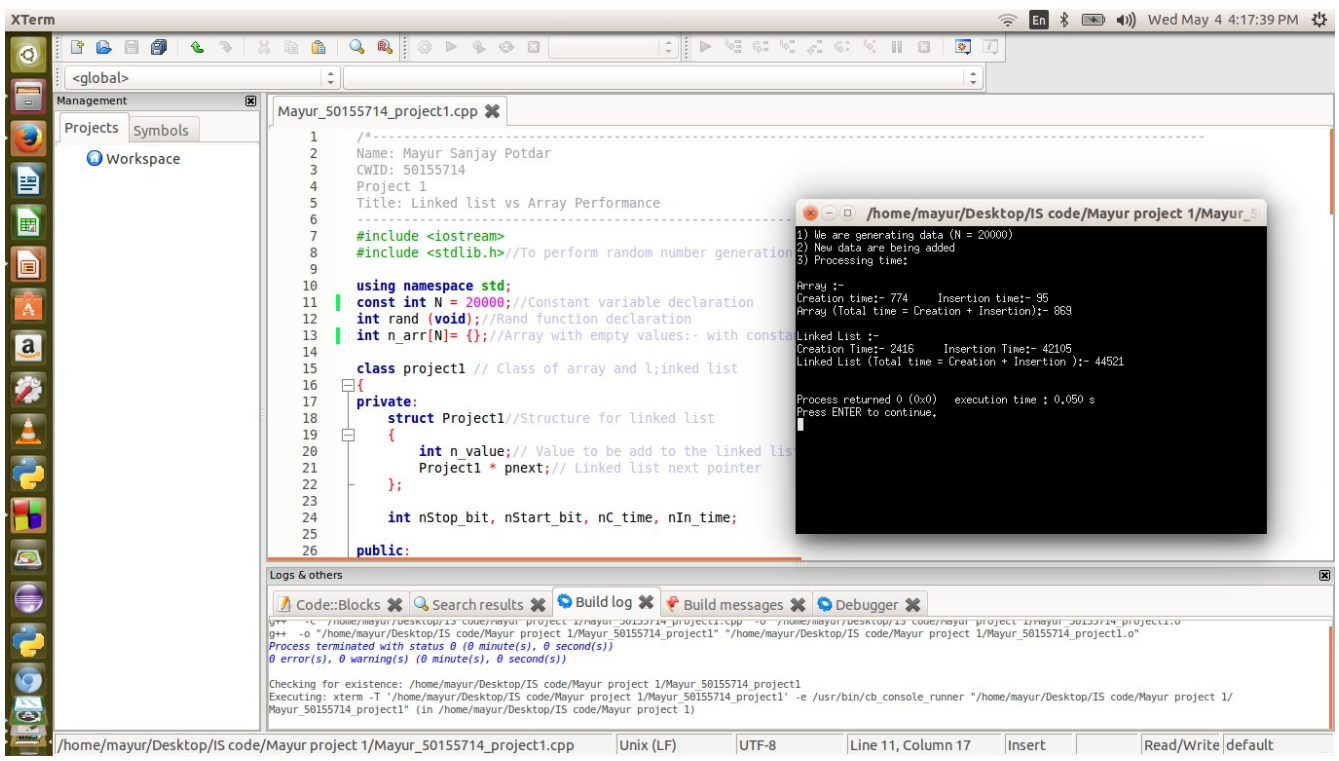


Figure 2. 5000 values for the comparison

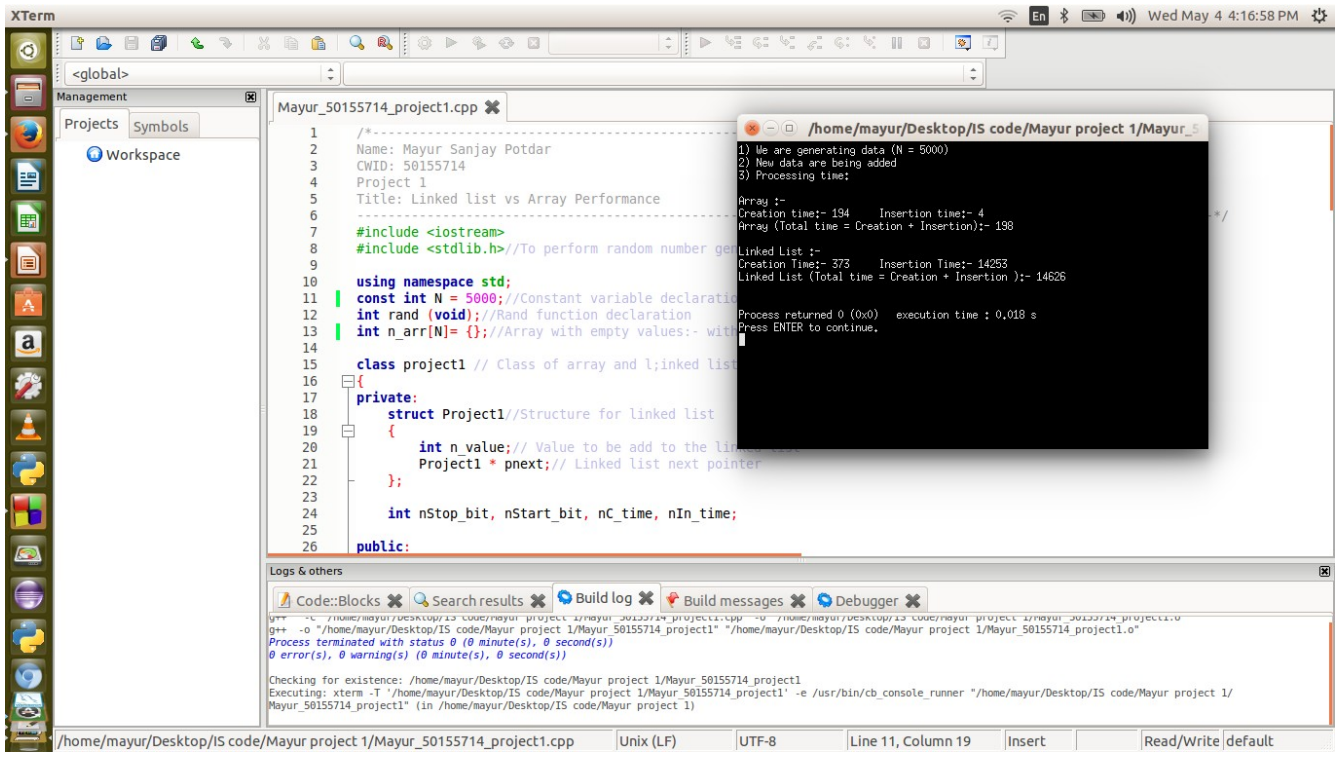


Figure3. 1000 values for the comparison

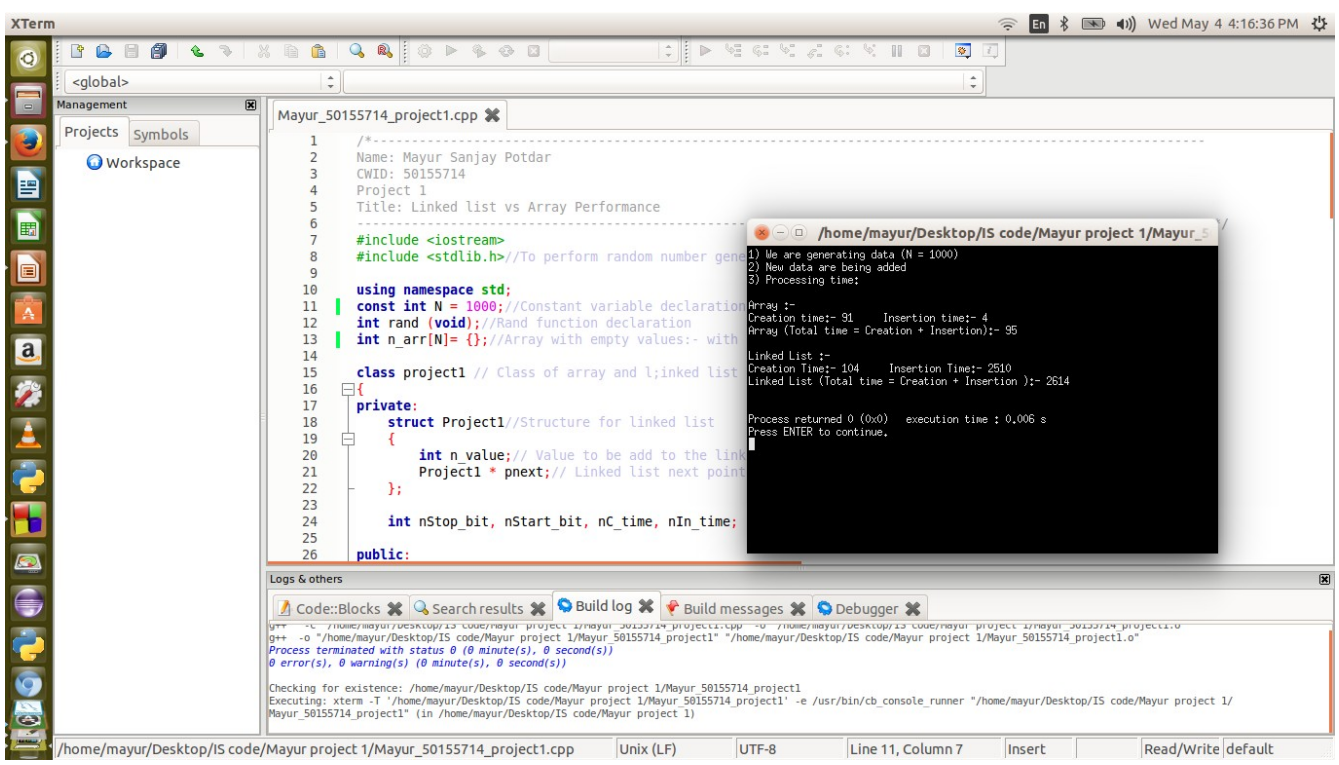


Figure 4. 100 values for the comparison

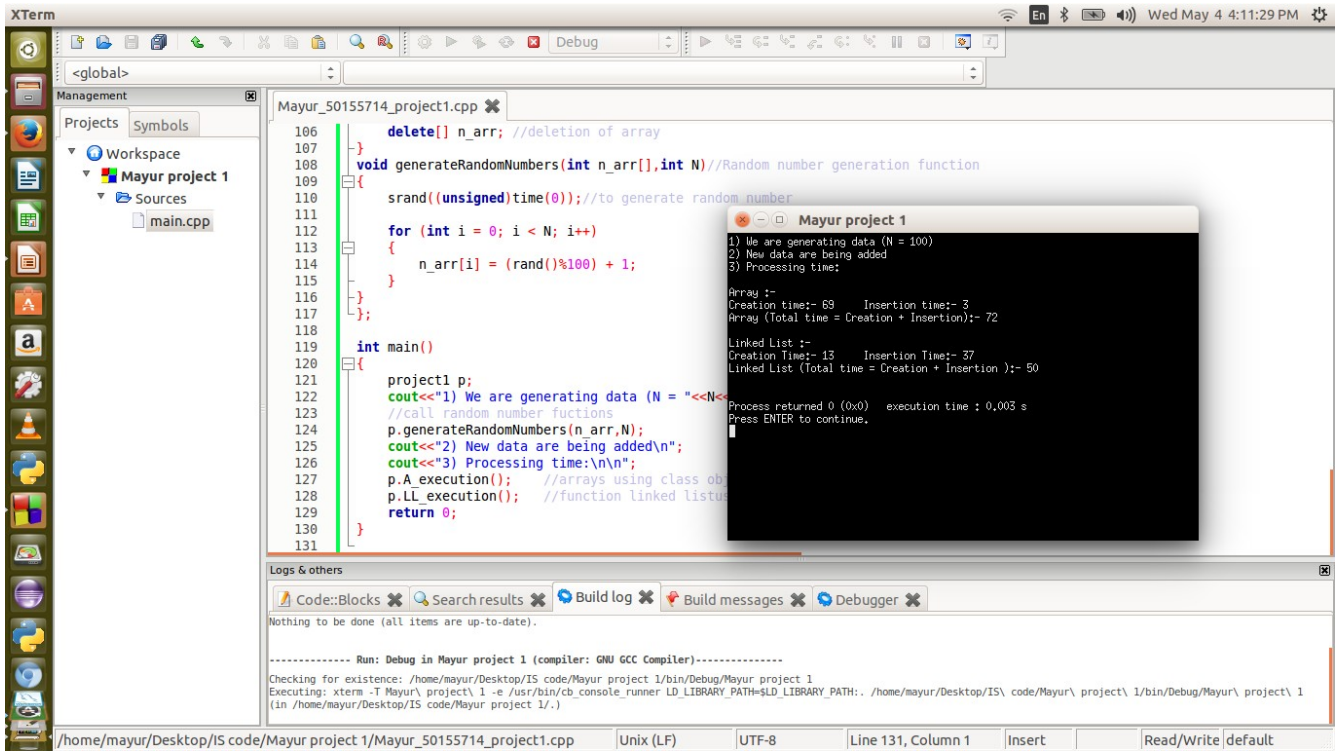
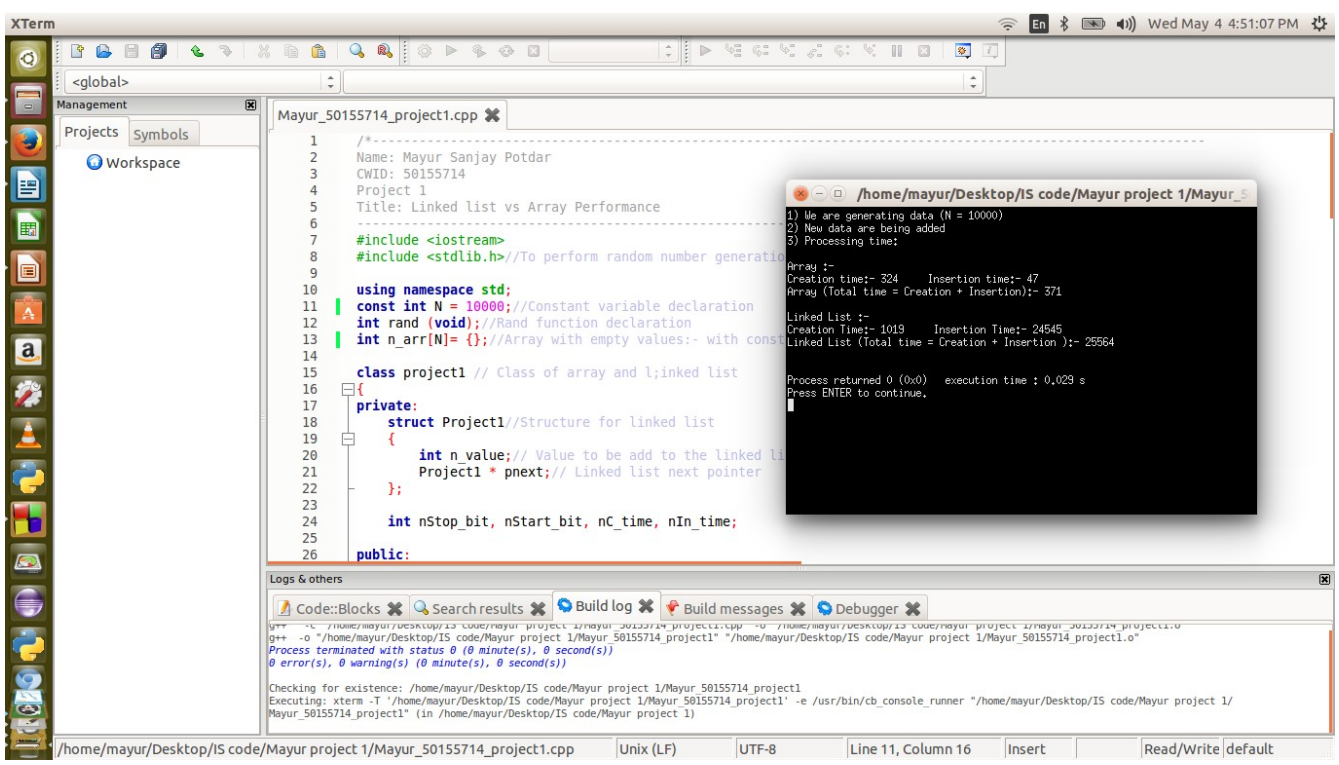


Figure 4. 10000 values for the comparison



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

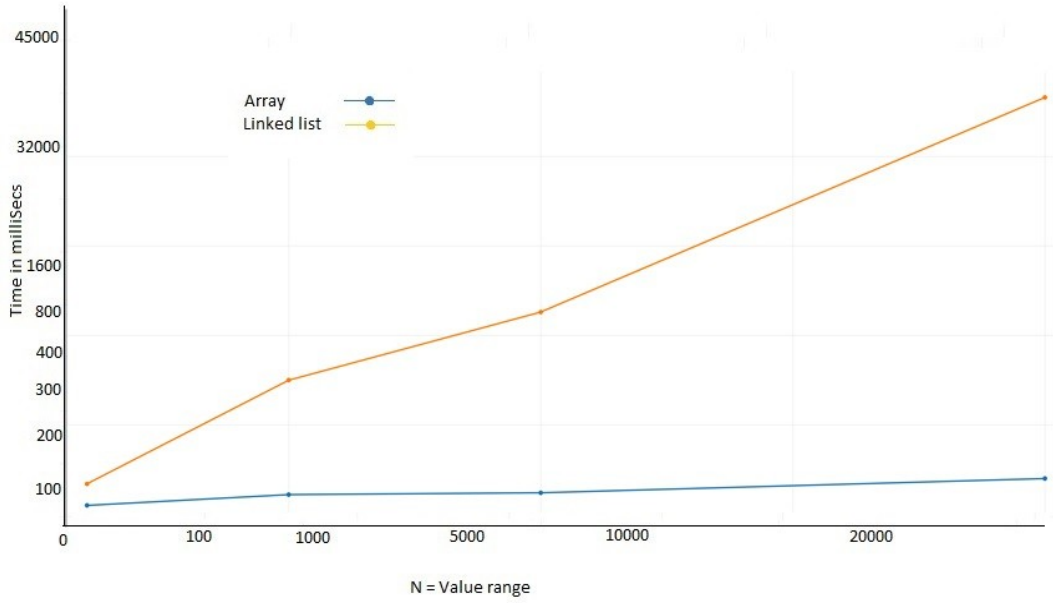


Figure 6. Value Insertion of array and linked list

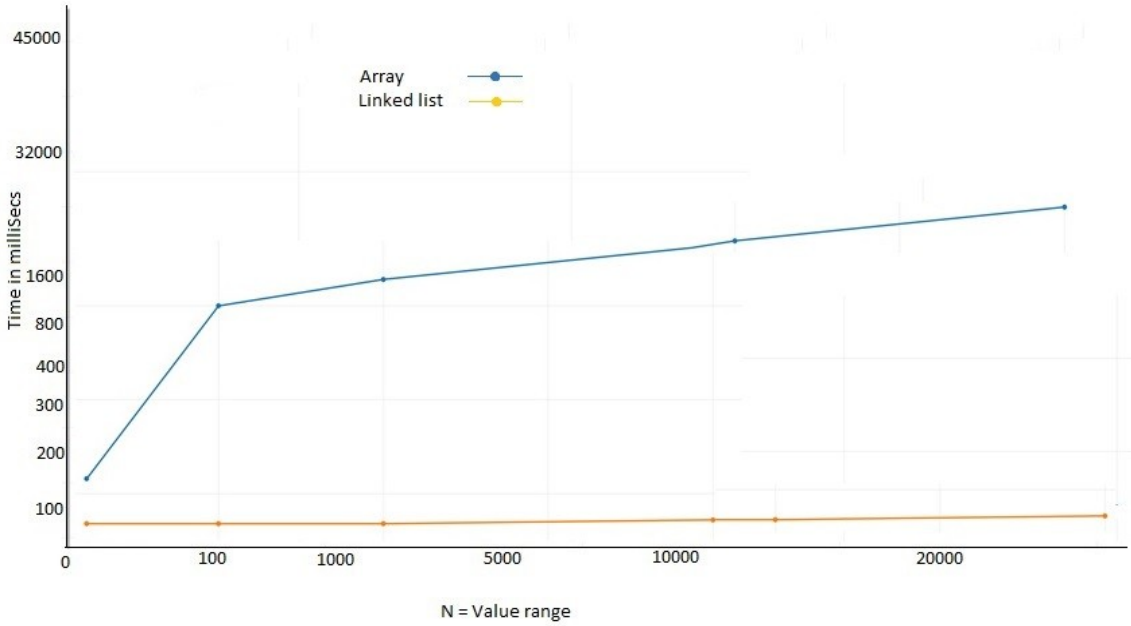
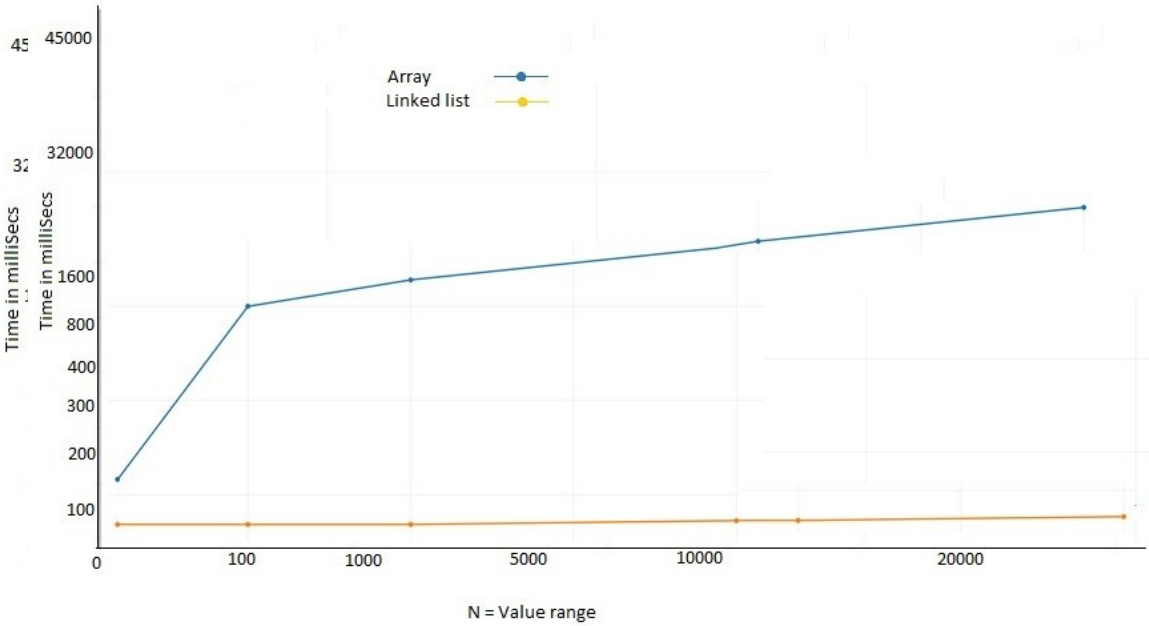


Figure 7. Final Result of array and linked list



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.