HACETTEPE UNIVERSITY ENGINEERİNG DEPARTMENT
COMPUTER ENGINEERİNG

LESSON

BBM204 PROGRAMMING LAB.

-

NAME: TAHA BAŞKAK

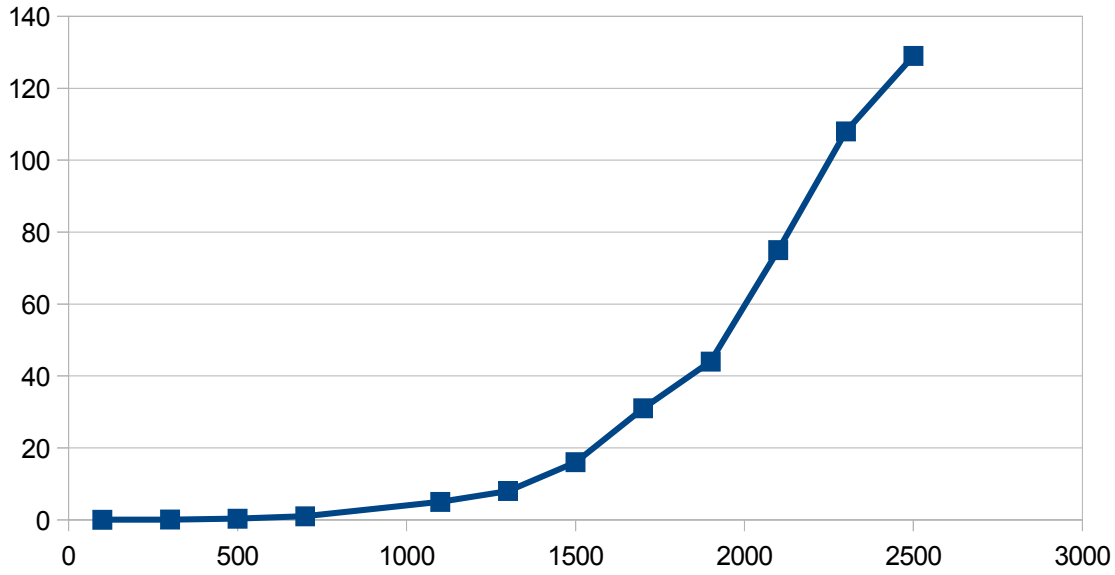
NUMBER: 21228104

ALGORİTHM'S TİME

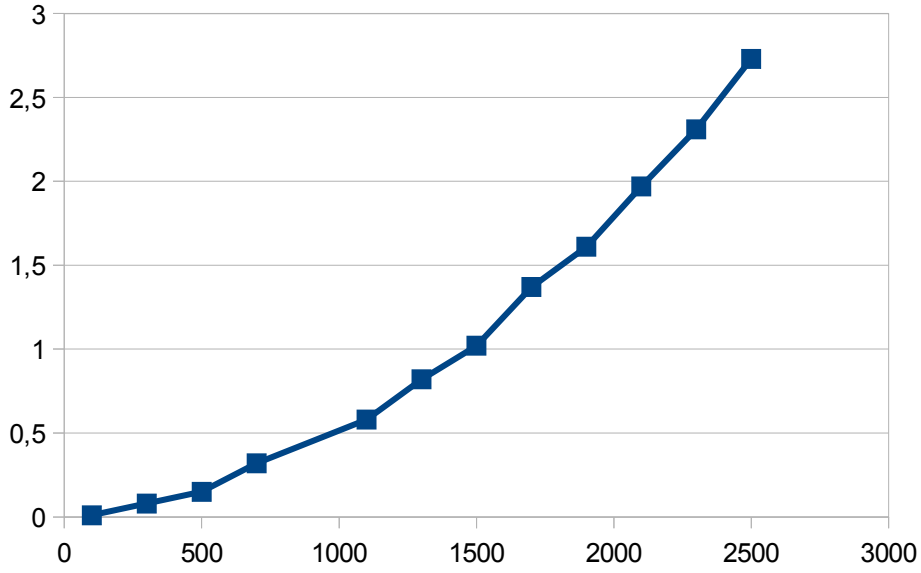
	100	300	500	700	1100	1300	1500	1700	1900	2100	2300	2500	
Matrix Mul.	0,01	0,07	0,3	1	5	8	16	31	44	75	108	129	s
Bubble Sort	0.01	0.08	0.15	0.32	0.58	0,82	1,02	1,37	1,61	1,97	2,31	2,73	ms
Finding Max Element	0.13	0.32	0.30	0.37	0.56	0.65	0.74	0.84	0.92	1,01	1,14	1,2	10^-6 s
Merge Sort	0.84	2,48	3,73	4,38	7,75	9,44	10,93	12,87	14,2	15,39	16,19	17,94	10^-6 s
Binary Search	0,76	0,92	0,97	1,02	1,11	1,13	1,14	1,12	1,15	1,19	1,21	1,25	10^-6 s

Thats algorithms is very speed. But Algorithms's time complexity changing some state. Because of that thats algorithm have diffirent memory requirements and time complexity. Binary Search Algorithm searching quickly , bubble sort and merge sort algorithms are sorting unsorted array , finding max. element is finding search number in unsorted array.

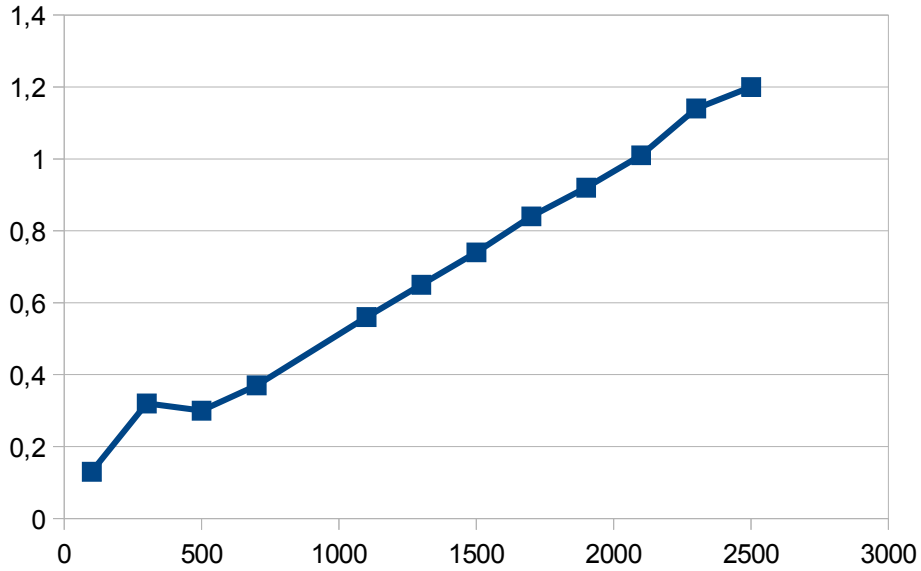
Matrix Multiplication ALGORİTHM (ms)



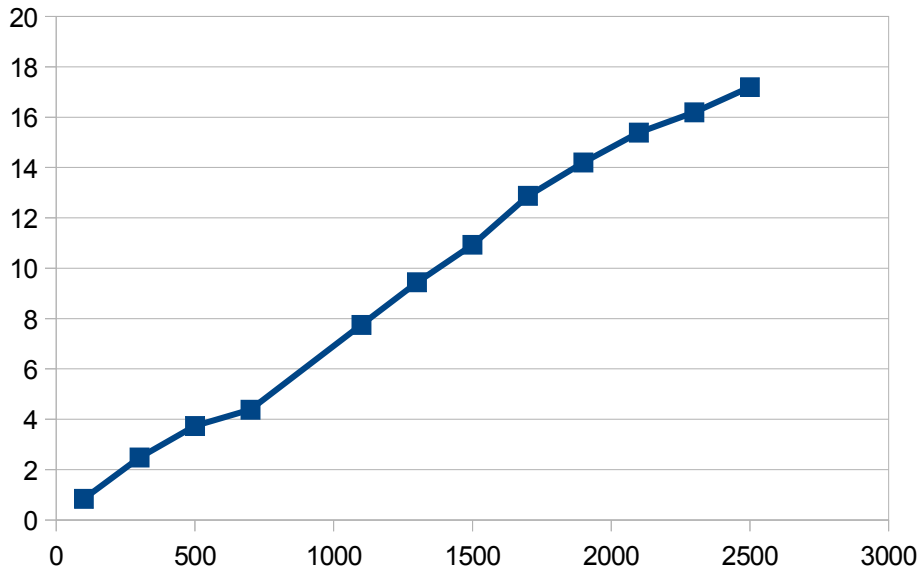
BUBBLE SORT ALGORİTHM (ms)



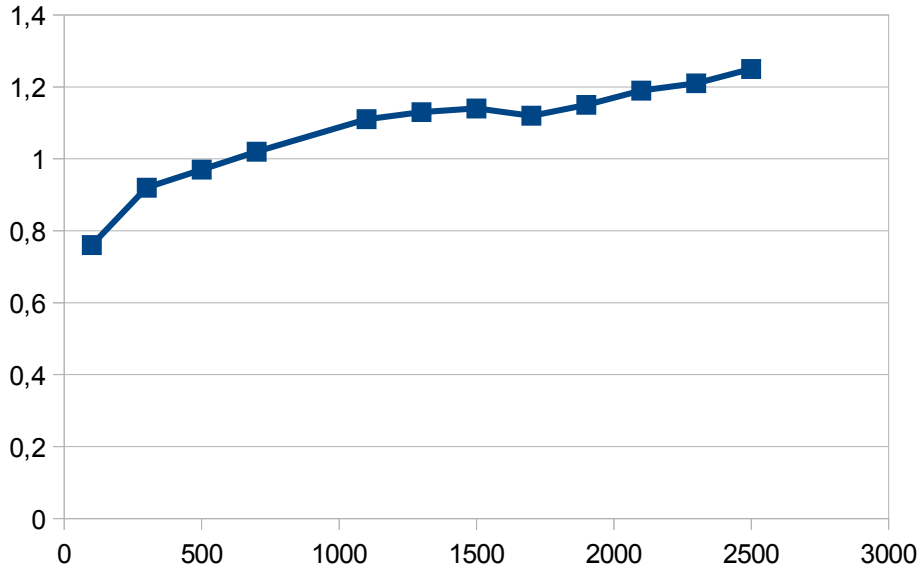
FİNDİNG MAXIMUM ELEMENT ALGORİTHM (10 ^-6 s)



MERGE SORT ALGORİTHM (10 ^ -6 s)



BİNARY SEARCH ALGORİTHM (10 ^ -6)



ALGORİTHMS ANALYSİS

Matrix Multiplication

```
for(i=0;i<n;i++){                                          c1          n
  for(j=0;j<n;j++){                                        c2          n*n
    for(k=0;k<n;k++){                                      c3          n*n*n
      sum = sum + (matrixOne[i][k] * matrixTwo[k][j]);     c4          n*n*n
    }
    multiplicationMatrix[i][j] = sum;                      c5          n*n
    sum =0;                                                c6          n*n
  }
}
```

	Unit Cost	Total Cost

Total Cost : c1*n +c2*n*n +c3*n*n*n +c4*n*n*n +c5*n*n + c6*n*n

Time complexity : O(n^3)

Bubble Sort

```
for(i=0;i<(n-1);i++){                                  c1          n
  for(j=0;j<(n-i-1);j++){                              c2          n*(n-1)
    if(bubbleArray[j] > bubbleArray[j+1]){             c3          n*(n-1)
      swap = bubbleArray[j];                           c4          n*(n-1)
      bubbleArray[j] = bubbleArray[j+1];               c5          n*(n-1)
      bubbleArray[j+1] = swap;                         c6          n*(n-1)
    }
  }
}
```

	Unit Cost	Total Cost

Total Cost : c1*n +c2*n*(n-1) +c3*n*(n-1) +c4*n*(n-1) +c5*n*(n-1) +c6*n*(n-1)

Time complexity : O(n^2)

Finding Maximum Element

```
maxElement = maxElementArray[0];                       c1          1
for(i=0;i<n;i++){                                      c2          n
  if(maxElementArray[i] > maxElement){                 c3          n
      maxElement = maxElementArray[i];                 c4          n
      }
}
```

	Unit Cost	Total Cost

Total Cost : c1 +c2*n +c3*n +c4*n

Time complexity : O(n)

Merge Sort

```
int i , leftSize, rightSize, mid;                          Unit Cost: c1   Total Cost: 1

if(size <2)                                                c2              1
  return;                                                  c3              1
mid = (size/2) ;                                           c4              1
leftSize = mid ;                                           c5              1
rightSize = size - mid;                                    c6              1
int leftMergeArray[] = new int[leftSize];                  c7              1
int rightMergeArray[] = new int[rightSize];                c8              1
for(i=0;i<mid;i++)                                         c9              n/2
  leftMergeArray[i] = mergeArray[i];                       c10             n/2
  for(i=mid;i<size;i++)                                    c11             (n/2)*(n/2)
    rightMergeArray[i - mid] = mergeArray[i];              c12             (n/2)*(n/2)
mergeSort(leftMergeArray,leftSize);                        c13             1
mergeSort(rightMergeArray,rightSize);                      c14             1
merge(leftMergeArray,rightMergeArray,mergeArray);          c15             1
```

Total Cost : c1 +c2 +c3 +c4 +c5 +c6 +c7+ c8 +c9*(n/2) +c10*(n/2) +c11*(n/2)*(n/2) +c12*(n/2)*(n/2) +c13 +c14 +c15

Time complexity : O(n^2)

Merge

```
int i=0,j=0,k=0;                                           Unit Cost: c1   Total Cost: 1
int leftMergeSize = leftMergeArray.length;                 c2              1
int rightMergeSize = rightMergeArray.length;               c3              1

while(i < leftMergeSize && j < rightMergeSize){            c4              n/2
  if(leftMergeArray[i] <= rightMergeArray[j]){             c5              n/2
    mergeArray[k] = leftMergeArray[i];                     c6              n/2
    i++;                                                   c7              n/2
    k++;                                                   c8              n/2
  }
  else if(leftMergeArray[i] > rightMergeArray[j]){         c9              n/2
    mergeArray[k] = rightMergeArray[j];                    c10             n/2
    j++;                                                   c11             n/2
    k++;                                                   c12             n/2
  }
}
while(i < leftMergeSize){                                  c13             n/2
  mergeArray[k] = leftMergeArray[i];                       c14             n/2
  k++;                                                     c15             n/2
  i++;                                                     c16             n/2
}
while(j < rightMergeSize){                                 c17             n/2
  mergeArray[k] = rightMergeArray[j];                      c18             n/2
  j++;                                                     c19             n/2
  k++;                                                     c20             n/2
}
return mergeArray;                                         c21             1
```

Total Cost : c1 +c2 +c3 +c4*(n/2) +c5*(n/2) +c6*(n/2) +c7*(n/2) + c8*(n/2) +c9*(n/2) +c10*(n/2) +c11*(n/2) +c12*(n/2) +c13*(n/2) +c14*(n/2) +c15*(n/2) +c16*(n/2) +c17*(n/2) +c18*(n/2) +c19*(n/2) +c20*(n/2) +c21

Time complexity : O(n)

Binary Search

```
while(first <= last){                                   c1          n
  if(binaryArray[middle] < searchNumber){               c2          n
    first = middle +1;                                  c3          n
  }
  else if( binaryArray[middle] == searchNumber){        c4          1
    System.out.println(searchNumber +                   c5          1
       " founded. Position is"
       + (middle +1) + ".\n" );
    break;                                              c6          1
  }
  else{                                                 c7          n
    last = middle -1;                                   c8          n
  }
  middle = (first + last )/2;                           c9          n
}
if(first > last){                                       c10         1
  System.out.println("Search number not found.\n");     c11         1
}
```

Total Cost : c1*n +c2*n +c3*n +c4 +c5 +c6 +c7*n +c8*n +c9*n +10 +c11
Time complexity : O(n)

REFERENCES

- http://www.javapractices.com/topic/TopicAction.do?Id=62
- http://stackoverflow.com/questions/10820033/make-a-simple-timer-in-java
- http://stackoverflow.com/questions/17623876/matrix-multiplication-using-arrays
- http://www.programmingsimplified.com/
- http://codereview.stackexchange.com/questions/64711/merge-sort-an-integer-array
- https://www.youtube.com/playlist?list=PL3NeFQ6er_3hSfYOBSn_E43meFFjQ3I0T