Tutorial on
Binary Search
Ternary Search
Searching/Sorting on strings

23rd May 2023

Q1. Which of the following is a prerequisite for applying binary search algorithm?

a) The array must be sorted in ascending order.

b) The array must be sorted in descending order.

c) The array can be in any order.

d) The array must have equal elements.

Q1. Which of the following is a prerequisite for applying binary search algorithm?

✓ a) The array must be sorted in ascending order.

b) The array must be sorted in descending order.

c) The array can be in any order.

d) The array must have equal elements.

Answer: a

Binary search relies on the property that elements to the left of the mid-point in a sorted array are smaller, and elements to the right are larger.

Q2. Which of the following is the correct implementation of the binary search algorithm in C?

a)

```
public static int iterative(int arr[], int key)
{ int low = 0;    int mid = 0;    int high = arr.length-1;
  while(low <= high) {
    mid = low + (high + low)/2;
    if(arr[mid] == key) {
      return mid;
    }
    else if(arr[mid] < key) {
      low = mid - 1;
    }
    else {
      high = mid + 1;
    }
  }
  return -1;
}
```

b)

```
public static int iterative(int arr[], int key)
{ int low = 0;    int mid = 0;    int high = arr.length-1;
  while(low <= high) {
    mid = low + (high - low)/2;
    if(arr[mid] == key) {
      return mid;
    }
    else if(arr[mid] < key) {
      low = mid + 1;
    }
    else {
      high = mid - 1;
   }
  }
  return -1;
}
```

c)

```
public static int iterative(int arr[], int key)
{ int low = 0;    int mid = 0;    int high = arr.length-1;
  while(low <= high) {
    mid = low + (high + low)/2;
    if(arr[mid] == key) {
      return mid;
    }
    else if(arr[mid] < key) {
      low = mid + 1;
    }
    else {
      high = mid - 1;
    }
  }
  return -1;
}
```

d)

```
public static int iterative(int arr[], int key)
{ int low = 0;    int mid = 0;    int high = arr.length-1;
  while(low <= high) {
    mid = low + (high - low)/2;
    if(arr[mid] == key) {
      return mid;
    }
    else if(arr[mid] < key) {
      low = mid - 1;
    }
    else {
      high = mid + 1;
    }
  }
  return -1;
}
```

Q2. Which of the following is the correct implementation of the binary search algorithm in C?

a)

```
public static int iterative(int arr[], int key)
{ int low = 0;   int mid = 0;   int high = arr.length-1;
  while(low <= high) {
    mid = low + (high + low)/2;
    if(arr[mid] == key) {
      return mid;
    }
    else if(arr[mid] < key) {
      low = mid - 1;
    }
    else {
      high = mid + 1;
    }
  }
  return -1;
}
```

//mid = low + (low+high)/2 is incorrect.

//low=mid-1 is incorrect.

✓b)

```
public static int iterative(int arr[], int key)
{ int low = 0;   int mid = 0;   int high = arr.length-1;
  while(low <= high) {
    mid = low + (high - low)/2;
    if(arr[mid] == key) {
      return mid;
    }
    else if(arr[mid] < key) {
      low = mid + 1;
    }
    else {
      high = mid - 1;
   }
  }
  return -1;
}
```

c)

```
public static int iterative(int arr[], int key)
{ int low = 0;   int mid = 0;   int high = arr.length-1;
  while(low <= high) {
    mid = low + (high + low)/2;
    if(arr[mid] == key) {
      return mid;
    }
    else if(arr[mid] < key) {
      low = mid + 1;
    }
    else {
      high = mid - 1;
    }
  }
  return -1;
}
```

//mid = low + (low+high)/2 is incorrect.

d)

```
public static int iterative(int arr[], int key)
{ int low = 0;   int mid = 0;   int high = arr.length-1;
  while(low <= high) {
    mid = low + (high - low)/2;
    if(arr[mid] == key) {
      return mid;
    }
    else if(arr[mid] < key) {
      low = mid - 1;
    }
    else {
      high = mid + 1;
    }
  }
  return -1;
}
```

//low=mid-1 is incorrect.

Q3. The minimum and maximum number of comparisons for a particular record among 32 sorted records through binary search method will be:

a) 1 and 32

b) 1 and 5

c) 1 and 4

d) 2 and 4

Q3. The minimum and maximum number of comparisons for a particular record among 32 sorted records through binary search method will be:

a) 1 and 32

✓ b) 1 and 5

c) 1 and 4

d) 2 and 4

Answer: b

0 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

(0+2)/2=0 (0+6)/2=3 (0+14)/2=7 (0+31)/2=15

(0+2)/2=1

Q4. Find the error(s).

```
int fun1(int arr[], int left, int right, int target) {
    if (right >= left) {
        int mid1 = left + (right - left) / 3;
        int mid2 = right - (right - left) / 3;

        if (arr[mid1] == target)    return mid1;

        if (arr[mid2] == target)    return mid2;

        if (target > arr[mid1]) {
            return fun1(arr, left, mid1 - 1, target);
        } else if (target < arr[mid2]) {
            return fun1(arr, mid2 + 1, right, target);
        } else {
            return fun1(arr, mid1 + 1, mid2 - 1, target);
        }
    }
    return -1;
}
```

Q4. Find the error(s).

```
int fun1(int arr[], int left, int right, int target) {
    if (right >= left) {
        int mid1 = left + (right - left) / 3;
        int mid2 = right - (right - left) / 3;

        if (arr[mid1] == target)    return mid1;

        if (arr[mid2] == target)    return mid2;

        if (target < arr[mid1]) {
            return fun1(arr, left, mid1 - 1, target);
        } else if (target > arr[mid2]) {
            return fun1(arr, mid2 + 1, right, target);
        } else {
            return fun1(arr, mid1 + 1, mid2 - 1, target);
        }
    }
    return -1;
}
```

Answer: (highlighted in green)

What does the code implement?

```
int fun1(int arr[], int left, int right, int target) {
    if (right >= left) {
        int mid1 = left + (right - left) / 3;
        int mid2 = right - (right - left) / 3;

        if (arr[mid1] == target)    return mid1;

        if (arr[mid2] == target)    return mid2;

        if (target < arr[mid1]) {
            return fun1(arr, left, mid1 - 1, target);
        } else if (target > arr[mid2]) {
            return fun1(arr, mid2 + 1, right, target);
        } else {
            return fun1(arr, mid1 + 1, mid2 - 1, target);
        }
    }
    return -1;
}
```

What does the code implement?

```
int fun1(int arr[], int left, int right, int target) {
    if (right >= left) {
        int mid1 = left + (right - left) / 3;
        int mid2 = right - (right - left) / 3;

        if (arr[mid1] == target)    return mid1;

        if (arr[mid2] == target)    return mid2;

        if (target < arr[mid1]) {
            return fun1(arr, left, mid1 - 1, target);
        } else if (target > arr[mid2]) {
            return fun1(arr, mid2 + 1, right, target);
        } else {
            return fun1(arr, mid1 + 1, mid2 - 1, target);
        }
    }
    return -1;
}
```

Answer: Ternary Search

Ternary search is a divide-and-conquer search algorithm that splits the search space into three parts. It repeatedly compares the target value with two midpoints, narrowing down the search interval.

Q5. Ternary search is more efficient than binary search when:

a) The array is unsorted

b) The array has a small size

c) The array has duplicate elements

d) The array is uniformly distributed

Q5. Ternary search is more efficient than binary search when:

a) The array is unsorted

b) The array has a small size

c) The array has duplicate elements

✓ d) The array is uniformly distributed

Answer: d

Ternary search divides the array into three parts, comparing the target value with two splitting points. It chooses which part of the array to search based on the comparisons. By dividing the search space into three equal parts, ternary search can potentially reduce the search space faster compared to binary search when the array is uniformly distributed.

Therefore, when the elements in the array are uniformly distributed, ternary search can provide more efficient search performance compared to binary search.

Q6. What will be the output of the following C code?

```
const char str1[10]="Helloworld";
const char str2[10] = "world";
char *mat;
mat = strstr(str1, str2);
printf("The substring is:%s\n", mat);
```

a) The substring is:world

b) The substring is:Hello

c) The substring is:Helloworld

d) error in the code

Q6. What will be the output of the following C code?

```
const char str1[10]="Helloworld";
const char str2[10] = "world";
char *mat;
mat = strstr(str1, str2);
printf("The substring is:%s\n", mat);
```

✓ a) The substring is:world

b) The substring is:Hello

c) The substring is:Helloworld

d) error in the code

Answer: a

Explanation: The C library function

char *strstr(const char *str1, const char *str2)
is used to find the first occurrence of the substring str2 in the string str1. The terminating '\0' characters are not compared.

Q7. What will be the output of the following C code?

```c
#include<stdio.h>
#include<string.h>

main( ) {
    char *str1 ="United" ;
    char *str2 ="it" ;
    char *str3 ;
    str3 = strstr(str1, str2);
    printf ( "\n%s", str3 ) ;
}
```

a) United

b) it

c) ited

d) Unit

Q7. What will be the output of the following C code?

```c
#include<stdio.h>
#include<string.h>

main( ) {
    char *str1 ="United" ;
    char *str2 ="it" ;
    char *str3 ;
    str3 = strstr(str1, str2);
    printf ( "\n%s", str3 ) ;
}
```

a) United

b) it

✓ c) ited

d) Unit

Answer: c

The strstr() function returns pointer to the first occurrence of the matched string in the given string. It is used to return substring from first match till the last character.

Q8. What is the output of the above code?

```c
#include <stdio.h>

int main() {
    char str[] = "Hello";
    char* ptr = str;
    printf("%s ", ptr++);
    printf("%s ", ptr);
    return 0;
}
```

a) ello ello

b) Hello

c) Hello ello

d) Ello Hello

Q8. What is the output of the above code?

```c
#include <stdio.h>

int main() {
    char str[] = "Hello";
    char* ptr = str;
    printf("%s ", ptr++);
    printf("%s ", ptr);
    return 0;
}
```

a) ello ello

b) Hello

✓ c) Hello ello

d) Ello Hello

Answer: c

The first printf statement prints the string pointed to by 'ptr' ("Hello") and then increments the pointer. The second printf statement prints the updated pointer value ("ello").

Q9. What is the output of the above code?

```c
#include <stdio.h>
#include <string.h>

int main() {
    char strArr[][100] = {"apple", "orange", "banana", "grape"};
    int n = sizeof(strArr) / sizeof(strArr[0]);

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (strcmp(strArr[i], strArr[j]) > 0) {
                char temp[100];
                strcpy(temp, strArr[i]);
                strcpy(strArr[i], strArr[j]);
                strcpy(strArr[j], temp);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        printf("%s ", strArr[i]);
    }

    return 0;
}
```

a) apple banana grape orange

b) apple banana orange grape

c) apple grape banana orange

d) orange banana apple grape

Q9. What is the output of the above code?

```c
#include <stdio.h>
#include <string.h>

int main() {
    char strArr[][100] = {"apple", "orange", "banana", "grape"};
    int n = sizeof(strArr) / sizeof(strArr[0]);

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (strcmp(strArr[i], strArr[j]) > 0) {
                char temp[100];
                strcpy(temp, strArr[i]);
                strcpy(strArr[i], strArr[j]);
                strcpy(strArr[j], temp);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        printf("%s ", strArr[i]);
    }

    return 0;
}
```

✓ a) apple banana grape orange

b) apple banana orange grape

c) apple grape banana orange

d) orange banana apple grape

Answer: a

Q10. What is the objective of the above code? (What is the code doing?)

```c
#include <stdio.h>
#include <string.h>

int main() {
    char strArr[][100] = {"apple", "orange", "banana", "grape"};
    int n = sizeof(strArr) / sizeof(strArr[0]);

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (strcmp(strArr[i], strArr[j]) > 0) {
                char temp[100];
                strcpy(temp, strArr[i]);
                strcpy(strArr[i], strArr[j]);
                strcpy(strArr[j], temp);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        printf("%s ", strArr[i]);
    }

    return 0;
}
```

Q10. What is the objective of the above code? (What is the code doing?)

```c
#include <stdio.h>
#include <string.h>

int main() {
    char strArr[][100] = {"apple", "orange", "banana", "grape"};
    int n = sizeof(strArr) / sizeof(strArr[0]);

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (strcmp(strArr[i], strArr[j]) > 0) {
                char temp[100];
                strcpy(temp, strArr[i]);
                strcpy(strArr[i], strArr[j]);
                strcpy(strArr[j], temp);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        printf("%s ", strArr[i]);
    }

    return 0;
}
```

Answer: Sorting

The code sorts the array of strings strArr in ascending order using the bubble sort algorithm. After sorting, the array will contain "apple", "banana", "grape", and "orange". The final loop prints each string in the sorted array, separated by a space.