Algoritmi
Pojam algoritma predstavlja temelj za razumijevanje računarstva općenito i stoga mu je u ovom dijelu posvećena posebna pažnja.

Neformalni algoritmi
Obzirom na naučeno gradivo trebali bi znati definirati algoritam koji, primjerice, pretvara različite formate numeričkih podataka, upravlja raspodjelom procesorskog vremena u višezadaćnom okruženju i sl. Algoritmom možemo opisati i izvođenje samog algoritma u stroju kao:

Sve dok se ne pojavi instrukcija za zaustavljanje, ponavljaj:
Dohvaćaj instrukciju
Dekodiraj instrukciju
Izvedi instrukciju

Algoritmi nisu ograničeni samo na tehničke aktivnosti. Pomoću algoritma možemo opisati mađioničarski trik ili neku svakodnevnu aktivnost, primjerice čišćenje graška kao:

Uoči košaru sa neočišćenim graškom i praznu zdjelu
Sve dok ima graška u košari, radi:
Uzmi mahunu iz košare
Prelomi je da se na njoj pojavi otvor
Istresi grašak u zdjelu
Baci mahunu

Mnogi istraživači vjeruju da je bilo koja aktivnost ljudskog uma, uključujući zamišljanje, kreativnost i donošenje odluka, u osnovi rezultat izvođenja nekog od algoritama što je za posljedicu imalo razvoj znanstvene discipline poznate pod imenom Umjetna inteligencija.

Definicija
Da bi specificirali značenje i važnost algoritma za računarstvo, uvedimo formalnu definiciju:
Algoritam je uređeni skup jednoznačnih (nedvosmislenih), izvedivih koraka.
Analizirajmo definiciju detaljnije:
1. Kako je riječ o uređenom skupu, logično je da se koraci algoritma trebaju izvoditi određenim redoslijedom. Međutim to ne znači da se koraci algoritma uvijek izvode sekvencijalno, u nizu. Postoje i tzv. paralelni algoritmi koji sadrže takozvane «niti» (thread) koje izvodi pojedini procesor, pa se određeni koraci mogu izvoditi istovremeno, ovisno o tome kako se niti granaju i ovisno o tome kako pojedini procesori izvode svoj dio zadatka. Osim što algoritam može izvoditi procesor, I bistabilni sklopovi mogu izvoditi specifične algoritme, tako da svaka vrata izvode jedan korak u algoritmu. Ovdje su koraci izvođenja uvijek uvjetovani uzrokom i posljedicom, pošto se izlazi iz jednih vrata šire kao novi signal ekektroničkim sklopom.
2. Pojam "jednoznačan" (nedvosmislen) u okviru definicije podrazumijeva da za vrijeme izvođenja algoritma informacija o stanju procesa mora biti dovoljna da jednoznačno I potpuno definira akciju koja slijedi. Drugim riječima, izvođenje (napredovanje) algoritma ne smije se temeljiti na kreativnosti (procesora, osobe koja ga slijedi), nego mora biti rezultat slijeđenja uputa.

Izuzeci od formalne definicije:
Definicija podrazumijeva izvođenje konačnog procesa, procesa koji ide prema nekom cilju.

Stavljanjem naglaska na konačnost izvođenja, dotiču se pitanja kojima se i inače bavi teorija računarstva, tipa:
- Koje je krajnje ograničenje algoritama i koji su dosezi stroja općenito?
Na ovaj su način u okvirima računarstva razdvojeni problemi koji se mogu riješiti algoritamski od onih koji nadilaze mogućnosti algoritama i s tim u vezi, razgraničeni su procesi koji rezultiraju odgovorom na pitanja(e), od onih na koje odgovori ne postoje. Životno iskustvo, međutim, pokazuje da postoje smislene aplikacije (temeljene na odgovarajućim algoritmima) koje uključuju odvijanje nekakvog beskonačnog procesa, primjerice, nadziranje životnih funkcija hospitaliziranih pacijenata ili automatski pilot koji održava visinu letjelice. U pokušaju da algoritme koji prate takve procese "uklope" u definiciju, neki ih autori predstavljaju kao konačne algoritme koji se automatski ponavljaju (beskonačno) kada dođu do kraja. Na ruku činjenici da strogu, formalnu definiciju algoritama treba prihvatiti uz postojanje izuzetaka koji je potvrđuju, ide algoritam kojim dobivamo rezultat dijeljenja dvaju cijelih brojeva. U slučaju da se dobije periodički decimalni broj, inače uspješni algoritam ne daje konačni odgovor na postavljeno pitanje.

Apstraktna priroda algoritma

Važno je naglasiti da postoji razlika između algoritma i načina na koji ga predstavljamo. Slična analogija vrijedi i među pojmovima «priča» i «knjiga». Priča je u osnovi apstraktni pojam, a knjiga je fizička manifestacija priče. Ako je prevedemo i tiskamo na drugom jeziku, mijenja se način na koji smo je predstavili, ali je priča ostala ista.
Algoritam je uvijek apstraktniji od načina na koji ga predstavljamo, a načini mogu biti različiti, ovisno o tome kome je namijenjen. Primjerice, algoritam koji pretvara stupnjeve Celzijusa u Fahrenheit-e, može biti predstavljen na slijedeći način:

- formulom: F=(9/5)*C+32
- riječima: pomnoži temperaturu u stupnjevima Celzijusa sa (9/5) i produktu dodaj 32, ili
- sklopom.

Ovisno o upućenosti osobe u problematiku, algoritam predstavljamo sa manje ili više detalja. Želimo li algoritam pripremiti za izvođenje na računalu, dva su standardna načina pomoću kojeg ga predstavljamo. Riječ je o predstavljanju pomoću:
- primitiva i
- pseudokoda

Primitive

Za predstavljanje algoritma potrebna nam je nekakva forma slična jeziku. Ako ljudi jedan drugom objašnjavaju algoritam, onda mogu koristiti riječi običnog, govornog jezika ili jezik slika. Mana takvog načina komunikacije je mogućnost da određene riječi budu shvaćene drugačije zbog višeznačnosti koja inače karakterizira riječi prirodnog jezika ili zbog kompozicije riječi unutar rečenice koja se može shvatiti na različite načine.
Algoritam pripremljen za računalo bit će jasan ukoliko se koriste specifični oblici (tzv. primitive) za točno definiranu vrstu funkcije koju odgovarajući oblik predstavlja. Taj skup osnovnih oblika, zajedno sa skupom pravila koja definiraju kako se osnovni oblici mogu koristiti čine programski jezik. (50-tih i 60-tih bili su vrlo popularni tzv. blok dijagrami za predstavljanje algoritama, ali kako se često događalo da zbog nepreglednosti ispisa izmakne temeljni smisao algoritma, predstavljanje pomoću blok dijagrama je danas gotovo zanemareno i uglavnom zamijenjeno predstavljanjem pomoću pseudokoda). Svaka primitiva

sastoji se od dva osnovna dijela: sintakse i semantike. Sintaksa omogućava simboličku oznaku primitive, a semantika joj daje značenje. Primjerice, sintaksa za ZRAK su 4 slova, a semantika bi označavala plinovitu mješavinu koja okružuje svijet. Da bi definirali skup primitiva kojima predstavljamo algoritam na računalu, mogli bi odabrati naredbe iz instrukcijskog skupa, što naravno nije zgodno, pa se zato koriste primitive "viših (apstraktnijih) nivoa".

Pseudo kod

Zamijenimo li formalnu, strogu strukturu programskog jezika, manje formalnim sustavom označavanja, dobili smo sustav za predstavljanje algoritma poznat pod nazivom PSEUDO KOD pomoću kojeg ideje možemo izraziti (zapisati) na lakši i manje formalan način tijekom postupka razvijanja (smišljanja) samog algoritma. Jedan od načina predstavljanja algo. pseudokodom je izostavljanje pravila iz naredbi koja prate svaki formalni programski jezik. Ovaj se način obično koristi kada znamo koji ćemo programski jezik koristiti, pa je tako semi-sintaktička struktura kojom opisujemo algoritam velikim dijelom nalik, ali je manje formalna od programskog jezika. Želimo li ipak ponuditi formalni zapis za predstavljanje algoritma neovisan o programskom jeziku, tada formaliziramo oznake za ponavljajuće semantičke strukture i definiramo pseudo kod kao intuitivni način predstavljanja algoritma kod kojeg ipak postoje određena pravila zapisa. Semantičke strukture koje se izmjenjuju pri opisu algoritma i koje "pokrivamo" pseudokodom su slijedeće:

1. Pridruživanje opisnog imena odgovarajućim vrijednostima:
ime = varijabla

2. Strukture koje nude mogućnost odabira jedne od dvije (ili više) ponuđenih vrijednosti na temelju zadovoljenog uvijeta. Tipični primjeri takvih struktura mogu se definirati rečenicom: Ako promet robom raste, naručuje se uobičajena količina proizvoda, inače, uskladištena se roba nudi po nižoj cijeni koju lako prilagođavamo strukturi pseudokoda oblika:

If (uvjet) Then (akcija1)
Else (akcija2)

Pri čemu koristimo ključne riječi **if**, **then** i **else** da bi naznačili podstrukture u strukturi i koristimo zagrade da bi naznačili granice podstruktura. Na taj je način definirana sintaksa određene semantike prilagođena pseudokodu.

3. Semantička struktura koja udovoljava zahtijevu za izvođenjem određenih aktivnosti sve dok neki uvjet ne postane istinit. Primjer izražen rečenicom:
Sve dok sve karte nisu prodane, biljetarnica je otvorena.
Takve se izjave uklapaju u obrazac:

While (ima karata za prodaju) **do** (prodavati karte).

4. Pseudokod koristimo i da bi apstrahirali niz aktivnosti za neku drugu aplikaciju. U računarstvu se takav skup koraka označava različitim imenima: procedura, funkcija, potprogram, modul, a svaki ima, ovisno o aplikaciji, svoje specifične karakteristike i značenje. Izraženo pseudokodom, jedan bi takav skup mogli predstaviti slijedećom strukturom:

Procedure Pozdrav

```
Brojčanik = 3
While (Brojčanik>0) do
( print "hello" and
Brojčanik = Brojčanik-1
)
```

Cilj pseudokoda je izraziti algoritam na čitljiv i neformalan način. Zato je način kojim ga predstavljamo samo pomoć za predstavljanje ideja, a ne način da ideju ugušimo. Da bi sve skupa bilo čitljivije pseudokod pišemo sa komentarima koji naznačavaju i opisuju pojedine strukture.

Zadatak: Napišite pseudokod za Eulerov algoritam koji rezultira najvećim zajedničkim djeliteljem dvaju pozitivnih cijelih brojeva!

```
x = veći ulaz
y = manji ulaz
while (y nije 0) do
(Ostatak = ono što ostane pri cjelobrojnom dijeljenju x i y
x = y
y = Ostatak
)
Rezultat = x
```

Otkrivanje algoritma

Razvijanje programa sastoji se od dvije aktivnosti:
- otkrivanje algoritma
- predstavljanje algoritma programom

Pomoću pseudokoda algoritam smo predstavili ne vodeći računa o tome kako se do njega došlo. Zapravo je puno teži , veći i zanimljiviji posao otkriti algoritam. I za to postoje odgovarajuće metode; razumijeti kako se algoritmi pronalaze ekvivalentno je razumijevanju kako se općenito rješavaju problemi. Problematika teorije riješavanja problema nije tipična samo za računarstvo, nego je sastavni dio gotovo svih znanstvenih disciplina. Zbog potrebe da se što efikasnije otkrije algoritam, računarci su tražili odgovore zajedno sa stručnjacima iz drugih područja koji su bili zainteresirani za pronalaženje tehnika kojima se općenito rješavaju problemi. Krajnji doseg tih nastojanja trebao je biti opet algoritam za rješavanje bilo kojeg problema, što je naravno nemoguće definirati, ali smanjimo li ambicije u dosezanju krajnjeg cilja, moguće je definirati neke korisne smjernice koje ubrzavaju put do rješenja. Kao vrlo korisna pokazala se receptura koju je 1945. definirao matematičar Polya (1887, Budapest-1985, Pao Alto, USA) sa ciljem usavršavanja rješavanja zadataka općenito, koja glasi:

1. Shvati problem
2. Donesi plan za riješavanje
3. Provedi plan
4. Provjeri točnost plana i procijeni koliko je primjenjiv na druge probleme

Polya-in koncept prenesen u kontekst razvijanja programa glasio bi:

1. Shvati problem
2. Smisli kako problem predstaviti algoritamski
3. Formuliraj algoritam
4. Provjeri točnost plana i procijeni koliko je primjenjiv na druge probleme

Iskustvo pokazuje da Polyain koncept ne treba shvatiti kao recepturu koju treba strogo poštivati, nego više kao aktivnosti koje slijedimo u nekoj fazi rješavanja problema. Konkretno, ljudi koji se bave programiranjem jako često formuliraju dijelove algoritma i prije nego problem shvate u cijelosti. Ukoliko se postavljena strategija ne pokaže dobrom, autor obično uvidi greške koje mu pomažu da problem bolje sagleda (shvati), pa u drugom pokušaju najčešće ima više uspjeha. Idealno bi bilo da tijekom riješavanja zadataka nema uzaludno potrošenog vremena, nego da nas svaki korak primiče cilju. Ukoliko se razvijaju jako veliki softwerski sustavi, tada pogreška uočena tijekom 4. faze može značiti jako veliki trošak i upravo je zadatak softwerskog inženjerstva da u što većoj mjeri eliminiraju mogućnost nastajanja takvih situacija. Jedan od načina je da globalne smjernice za razvoj softwerskih modula postavljaju ljudi koji imaju uvid u težinu i način rješavanja problema.

S druge strane, praksa nam govori o problemima koji se mogu riješiti algoritamski, a da dostupni podaci naizgled ne predstavljaju kvalitetnu osnovu koja bi nas mogla dovesti do cilja. Sama činjenica da se konkretno riješenje za problem ne nazire, uvijek u sebi krije mogućnost da problem nije dobro shvaćen.
Tipičan primjer koji dobro ilustrira gore navedene mogućnosti predstavljen je pričom: Osoba A treba pogoditi koliko imaju godina djeca osobe B. Da bi joj olakšala posao, osoba B kaže osobi A da ima troje djece i da je produkt njihovih godina 36. Nakon što je promislila, osoba A, koja je inače jako dobar programer, kaže da bi joj bila nužna barem još jedna dodatna informacija, nakon čega osoba B uzvrati da može pretpostaviti da joj je poznata i suma godina njene djece. Kako je A rekla da joj na temelju danih informacija nitko ne može ponuditi točan odgovor čak ni kad bi se točno znalo koliko iznosi suma njihovih godina, B je rekla neka onda vodi računa o tome da joj najstariji sin jako dobro svira klavir. Zadovoljna pristiglim informacijama, A precizno izvijesti osobu B da zna kako ima dvogodišnje blizance i da je sigurno ponosna na devetogodišnjeg sina koji svira klavir!?

Kako je osoba A pronašla točno riješenje, ako je druga informacija bila nepotpuna (nije rekla kolika je točno suma godina), a treća se čini besmislenom.
U trenutku kada je B rekla da je produkt godina njene djece 36, A je napravila listu svih mogućih kombinacija:

(1 1 36)
(1 2 18)
(1 3 12)
(1 4 9)
(1 6 6)
(2 2 9)
(2 3 6)
(3 3 4)

nakon čega je zatražila dodatnu informaciju. Uz pretpostavku da je poznata i suma njihovih godina lista bi poprimila slijedeći oblik:

(1 1 36 38)

(1 2 18 21)
(1 3 12 16)
(1 4 9 14)
(1 6 6 13)
(2 2 9 13)
(2 3 6 11)
(3 3 4 10)

Iz oblika liste, jasan je i smisao odgovora osobe A, jer dvije liste imaju isti iznos za sumu godina. Iako treća informacija zvuči sasvim besmisleno promatramo li zadatak kao problem rješavanja tri jednadžbe sa tri nepoznanice, ipak informacija osobe B, rješenju konkretnog problema daje potpuni smisao, jer razdvaja liste sa istim odgovorima.
Ovaj primjer jako dobro ocrtava karakteristike sličnih koje do kraja ne možemo razumijeti, ako ih ne pokušamo prije riješiti, zbog čega zapravo ne možemo definirati univerzalan i sistematičan put dolaska do cilja. Nije nebitna ni činjenica da većina programera prolazi kroz iskustvo koje se očituje «bljeskovima» tijekom kojih se dobiva ideja za riješavanje nekog problema od kojeg smo davno prije odustali. Psiholozi tumače pojavu kao djelovanje nesvjesnog koje je tijekom vremena savladalo problem i gurnulo rješenje u područje svjesnog. Na temelju svega toga jasno je da put do pronalaska rješenja umnogome ovisi o talentu i ambicijama onoga koji ga rješava i da mu treba pristupati više kao filozofskom pitanju, nego kao problemu čiji se koraci mogu točno definirati.

Primjer 2:
Prije utrke A, B, C i D dali su svoje prognoze o ishodu utrke:
- A je predvidio da će pobijediti B
- B je predvidio da će D biti posljednji
- C je predvidio da će A biti treći
- D je predvidio da je A dobro procijenio.

Samo je jedna progoza bila točna i nju je predvidio pobjednik. Kako je završila utrka?

A nije pobijedio jer je kazao da će pobijediti B;
B nije mogući pobjednik jer bi A imao točnu prognozu;
C je mogući pobjednik; A je u tom slučaju treći, B je posljednji (da bi prognoza B bila pogrešna), a D drugi, dakle točno rješenje je dobijeno metodom eliminacije i glasi: CDAB.

Rješenje smo relativno lako pronašli, ali zasigurno bi trebalo vremena da bismo naš put do cilja definirali kao metodu. Kako su se tom problematikom ljudi ipak bavili, postoji definirana metodologija pronalaženja riješenja definirana slijedećim postupcima, koje nam mogu barem skratiti vrijeme definiranja algoritma:

1. Unazadna metoda kod koje se kretanjem od rezultata nastoje pronaći odgovarajuće ulazne vrijednosti (tipična za slagalice od papira)
2. Dani se problem riješava po uzoru na sličan, lakši problem čije nam je riješenje poznato. Takav je pristup jako čest kod programiranja kada primjerice radimo algoritam za sortiranje tako da pravila primijenjujemo na konkretnu listu, a zatim nastojimo poopćiti riješenje.
3. Rješenju se približavamo postepeno, riješavanjem niza potproblema; zadatak rastavljamo na manje cjeline, od kojih je svaka lakše riješiva od samog problema.

Pri tom, postupak 3. karakterizira se kao tzv. TOP-DOWN postupak, kod kojeg primijenjena

metodologija propagira od općeg na specifično, a postupci 1. i 2. kao BOTTOM-UP postupci koji propagiraju od specifičnog na opće.
U praksi, problemi se obično rješavaju kombiniranjem oba postupka jer programeri najčešće raščlanjuju problem na manje cjeline (top-down) za koje intuitivno vjeruju (bottom-up) da će doprinijeti općem rezultatu.
Iako je put dolaska do cilja različit, pri čemo nam na raspolaganju stoje različiti alati, dobro je ipak, ako ne potpuno razumijeti, ono barem promisliti malo o problemu koji stoji pred nama. Ponekad je riješenje sadržano u samom pitanju, ne treba ni razumijevanje da bismo postavili algoritam niti algoritam da bismo shvatili problem; potrebno je shvatiti da problem ni ne postoji. Tvrdnja je ilustrirana primjerom:

Ako nam u trenutku kada pokrenemo brod koji stoji na rijeci i krenemo uzvodno, padne kapa u rijeku i počne se gibati nizvodno nošena strujom rijeke čija je brzina 3 km/h, koliko će nam trebati vremena da je ponovno ugledamo, vozimo li uzvodno 10 minuta brzinom od 4 km/h relativno u odnosu na rijeku?
U ovom slučaju, umjesto da zbrajamo i oduzimamo brzine i računamo prijeđeni put, jasno je da ako brod okrenemo trebamo isto vrijeme da dođemo do kape jer se oba objekta gibaju po istom mediju.
Iz svega ovoga može se zaključiti da je zadatak pronalaženja algoritma jedna vrsta umjetnosti u kojoj različiti ljudi različito uživaju i pridonose. Ipak, dobro je ovladati alatima koji se jako često koriste pri programiranju kako bi što lakše prepoznavali tipične probleme i na njih nebi trošili dragocjeno vrijeme pri otkrivanju algoritma. Takve tipične strukture su tzv. iterativna i rekurzivna struktura.

Iterativne strukture

Kod iterativnih se struktura, skup instrukcija ponavlja u petlji. Tipični algoritmi koji oriste te strukture su algoritam za sekvencijalno pretraživanje. Krenimo redom.

Algoritam za sekvencijalno pretraživanje

Cilj nam je pretražiti listu (pojmova, imena) u potrazi za određenim ciljem (target). Algoritam nam treba dati odgovor da li je traženi cilj na listi ili ne. Pri tom, pretpostavljamo da je lista složena nekim redom (primjerice, abecednim). Logično je zadatak riješiti tako da članove liste, redom, uspoređujemo sa vrijednošću cilja sve dok član sa liste ne postane jednak traženoj vrijednosti ili po abecednoj vrijednosti veći od ciljnog člana. U tom slučaju zaustavljamo pretraživanje i objavljujemo rezultat pretraživanja. Pseudokodom, algoritam bi mogli predstaviti:

```
Procedure Traži(Lista, CiljnaVrijednost)
If (Lista prazna)
Then
(Pretraživanje neispravno jer nema elemenata u listi)
Else
(Odaberi prvu vrijednost u Listi kao TestVrijednost;
While (CiljnaVrijednost>TestVrijednost i
Stoga ima elemenata u Listi koje treba ispitati)
Do (Odaberi slijedeću vrijednost sa Liste kao TestVrijednost.);
If (CiljnaVrijednost=TestVrijednost)
Then (Objavi da je rezultat pretraživanja pozitivan)
```

Else (Objavi da je rezultat pretraživanja negativan)
) end if

Zbog jednostavnosti algoritam je primjeren pretraživanju malih listi. U ovom algoritmu iterativna struktura predstavljena je formom petlje kod koje se skup instrukcija (tijelo petlje) ponavlja u skladu sa postavljenim uvjetima.

Forma petlje ima tri komponente:
- inicijalizacija početnog stanja
- testiranje trenutnog stanja sa traženim stanjem
- modificiranje trenutnog stanja prema traženom

Tipičan primjer:

Broj = 1
While (Broj!=6)
Broj = Broj+1

Postoje dva tipična oblika ponavljajućih struktura koje predstavljamo petljama (tzv. pretest loop i posttest loop), a u pseudokodu ih predstavljamo slijedećim formama:

- repeat (aktivnost) until (uvjet)
- while (uvjet) do (aktivnost)

Algoritmi za uređivanje lista

U ovom slučaju treba riješiti problem slaganja liste sa imenima prema abecednom redoslijedu. Pri tom listu treba presložiti unutar sebe, premještanjem za jedno mjesto koje se oslobađa kada ime sa tog mjesta želimo staviti na pravo mjesto u listi. Pretpostavka je da listu sastavljenu od samo jednog imena ne treba slagati, ali onu od dva već treba. Prema tome, ako promatramo listu sa slijedećim imenima:

- Frane
- Ana
- Dino
- Blaž
- Crni

Sa prvim imenom ne radimo ništa, a kada ime Ana želimo staviti na pravo mjesto u listi, zamišljamo kao da mjesto broj 2 (mjesto na kojem je bila Ana) ostaje upražnjeno, oslobođeno za prvo gornje ime iznad Ane koje je eventualno po abecednom redu iza nje. Kako je to zaista tako (Željko>Ana), Ana postaje prvim imenom na listi, a Željko postaje drugo ime. Nakon toga, slažemo ime Dino. Treće mjesto ostaje upražnjeno, Željko pada dolje, a Dino zauzima drugo mjesto. Procedura se ponavlja sve dok se ne naiđe na ime koje je po abecednom redu prije Dina, konkretno u ovom slučaju, to će biti Ana. Procedura se ponavlja sve dok ne dođemo do kraja liste. Dakle, izraženo pseudokodom, algoritam glasi:

Procedure Sortiraj (Lista)
N=2;
While (vrijednost N ne postane jednaka broju članova liste) do

Pomakni se do željenog mjesta na listi, ostavljajući to mjesto upražnjeno;
While (ima imena iznad praznine i abecedni rang im je veći od željenog)
Do (ime iznad pomakni u prazninu dolje, ostavljajući prazninu na mjestu na kojem se nalazilo)
Pomakni željeni ulaz na to mjesto;
N=N+1;
)

Rekurzivne strukture

Predstavljene su skupom instrukcija koje se ponavljaju u petlji, a među njima se nalazi i instrukcija koja označava sve instrukcije iz petlje. Interesantno je promotriti kako se prethodno opisani algoritam za pretraživanje liste može definirati pomoću ovakve strukture:

Procedure Traži (lista, CiljnaVrijednost)
If (lista prazna)
Then javi grešku
Else
Odaberi srednju vrijednost iz liste kao TestUlaz;
Usporedivši TestUlaz i CiljnuVrijednost, izvedi odgovarajući skup instrukcija, ovisno o rezultatu usporedbe (case);
Case 1: CiljnaVrijednost=TestUlaz
Javi rezultat da je ime na listi
Case 2: CiljnaVrijednost<TestUlaz
Pretraživanje se ponavlja samo za imena koja se nalaze iznad TestUlaza
Case 3: CiljnaVrijednost>TestUlaz
Pretraživanje se ponavlja samo za imena koja se nalaze ispod TestUlaza
) end if

Na ovaj način, u svakom se koraku pretraživanja broj imena na listi smanjuje na pola, pa se brže dolazi do rezultata. Algoritam pretraživanja liste pomoću rekurzivne strukture zove se algoritam binarnog pretraživanja..

Efikasnost i točnost

Obično nije problem dobiti točno rješenje pomoću algoritma predstavljenim programskim jezikom, ali o efikasnosti algoritma ovisi da li će pronađeno rješenje saživjeti u praksi. Ukoliko pretražujemo listu sa 30000 imena, tada algoritam sekvencijalnog pretraživanja u prosjeku pretraži 15000 imena dok ne dođe do cilja. Ako pretpostavimo da svako pretraživanje traje 10 ms, onda će nam za pronalazak imena u listi trebati (u prosjeku) 150 000 ms = 2.5 minuta. Kod binarnog pretraživanja broj koraka koji nas dovodi do ciljne vrijednosti iznosi ln (30000)=8, što znači da je ukupno vrijeme pretraživanja cca 80ms., s tim da treba voditi računa o tome da slogovi moraju biti složeni po nekom redu kako bismo algoritam uopće mogli primijeniti.

(Koliko se poveća vrijeme pretraživanja kod jednog, a koliko kod drugog algoritma ako se broj imena na listi udvostruči??- Upravo to, osjetljivost na porast broja elemenata u listi u ovom je slučaju mjera za efikasnost algoritma).

Nadalje, prema dobivenom rješenju treba biti kritičan. Čak i kada smo sigurni da je ono točno,

treba provjeriti je li riječ i o najboljem mogućem rješenju. (Priča o trgovačkom putniku koji hotel plaća karikama lanca-cilj je platiti sedam noćenja sa lančićem od sedam karika, tako da svaki dan platimo noćenje, a da lančić puknemo najmanji broj puta)